UNIVERSITETET I OSLO
Institutt for informatikk

Teknologi for sensordata-innsamling til mediespiller

Masteroppgave

Øyvind Nyborg Hauback

Vår 2011

Sammendrag

Denne oppgaven omhandler registrering av bevegelse til bruk i trådløse Aktiv Musikk-systemer, hvor bevegelse brukes til å lage og påvirke musikk. Jeg har vurdert ulik maskin- og programvare, valgt en sammensetning av disse, og testet hvordan denne fungerer til et slikt system.

Prototyper av Aktiv Musikk-systemer lages av mennesker med ulik bakgrunn. Noen er musikere, mens andre har bakgrunn innenfor elektronikk eller informatikk. For å kunne lage et velfungerende system, må man ha kunnskap og erfaring innenfor flere fagområder. En prototypeplattform med høy brukervennlighet, vil kunne gi mennesker med lite programmeringserfaring mulighet til å utvikle egne prototyper. Spørsmålet er om hastighet, stabilitet og tilkoblingsmuligheter er like gode, som på en plattform med dårligere brukervennlighet.

Etter å ha gjort en sammenligning av flere ulike plattformer med hovedvekt på brukervennlighet, kom jeg fram til et prototypekort som kalles CUI32. Dette kortet inneholder en mikrokontroller som kjører et fullverdig operativsystem kalt StickOS. Jeg utviklet videre et kretskort som kobles til CUI32 for å utnytte mulighetene StickOS har til å kommunisere trådløst.

Jeg testet flere oppsett med ulike sensorer og kom fram til at CUI32 med StickOS er raskt å komme i gang med, og kommuniserer med de fleste sensorer. På den andre siden er den trådløse overføringshastigheten lavere enn for tilsvarende systemer, omtrent $0,8 kbit/s$. Dette er ikke raskt nok til kontinuerlig bevegelsesanalyse som gjøres på bakgrunn av akselerometer- eller gyroskopsensorer. Allikevel er brukervennlighet, stabilitet og tilkoblingsmuligheter svært gode. Plattformen kan med fordel brukes til trådløse Aktiv Musikk-systemer, hvor overføringshastighet mellom sensor og mediespiller ikke er avgjørende. Dette vil gjelde Aktiv Musikk-systemer hvor musikken endres langsomt, eller systemer hvor analyse av sensordata kan gjøres før de blir overført. Slik at overføringsbehovet blir mindre.

For demonstrasjon av oppsettet som er brukt i denne oppgaven er det laget en film. Den er forklart og beskrevet i appendiks A.

Abstract

This thesis describes registration of motion for use in a wireless Active Music system, where movement is used to create and influence music. I have evaluated different hardware and software solutions, selected one combination of these and tested how it works in such a system.

Prototypes for use in Active Music systems are created by people with different backgrounds. Some are musicians, while others have a background in electronics or computer science. A prototype platform with high usability, will give those people with little programming experience the opportunity to develop their own prototypes. The question is whether the transmission speed, stability and connectivity are equally good, as for a platform with worse usability.

After making a comparison of several different platforms, with emphasis on usability, I chose to test a prototype card named CUI32. This card contains a microcontroller that runs the StickOS operating system. Furthermore, I developed a circuit board that connects to CUI32, in order to exploit the wireless capabilities of StickOS.

After testing several setups with different sensors, I came to the conclusion that CUI32 with StickOS is quick to get started with, and communicates with most sensors. On the other hand, the wireless transmission is not as fast as one could hope, about $0.8 kbit/s$. This is not fast enough for continuous motion analysis, made on the basis of accelerometers or gyroscopes. However, the usability, stability and connectivity are very good. The platform can be used for wireless Active Music systems, where transmission speed between the sensor and the media player is not essential. This will apply to Active Music systems where the music changes slowly, or systems where the sensor data can be analysed before being transmitted to the media player.

Forord

Denne oppgaven er en del av min mastergrad i Elektronikk og Datateknologi ved Universitetet i Oslo. Den er gjort for forskningsgruppen Robotikk og Intelligente Systemer (ROBIN) ved Institutt for Informatikk. Videre er den en del av et samarbeidsprosjekt mellom Institutt for Musikkvitenskap og Institutt for Informatikk. Prosjektet heter «Sensing Music-Related Actions» og har som mål å lære mer om sammenhenger mellom musikk og bevegelse, som kan brukes til å lage Aktiv Musikk-systemer og mediespillere.

Takk til mine veiledere Jim Tørresen og Alexander Refsum Jensenius for all hjelp under planlegging, implementering og skriving av denne masteroppgaven. Jeg vil også takke Yngve Hafting og andre ved Institutt for Informatikk som har bidratt og hjulpet meg videre. Til slutt, en stor takk til familie, venner og ikke minst Iseline, for all hjelp og støtte under arbeidet med denne oppgaven.

Øyvind Nyborg Hauback
Juni 2011

Innhold

Figurer

Tabeller

Kapittel 1

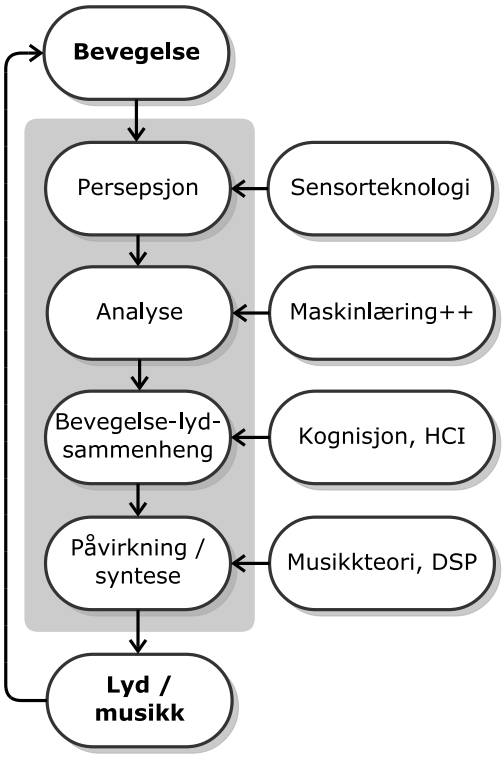
Introduksjon

1.1 Definisjoner

I dette kapittelet vil jeg gi en introduksjon til *Aktiv Musikk* og noen definisjoner knyttet til dette. Jeg vil beskrive motivasjonen som ligger bak oppgaven, ved hjelp av en del spørsmål og utfordringer knyttet til Aktiv Musikk-systemer. Til slutt vil jeg beskrive avgrensningen og oppbygningen av oppgaven.

Som regel er man passiv når man lytter til musikk. Med det mener jeg at med mindre man spiller et instrument, kan man kun lytte til musikken, uten å ha mulighet til å kontrollere den. Med dagens teknologi kan man ved hjelp av bevegelsessensorer og annen elektronikk ha en kontinuerlig kontroll på musikk ved hjelp av kroppsbevegelser. Dette kalles Aktiv Musikk og er nærmere beskrevet i avsnitt 2.2. [15]

For å kunne lage et Aktiv Musikksystem trenger man en *plattform* av maskin- og programvare som kan danne grunnlaget. Ulike sammenset-

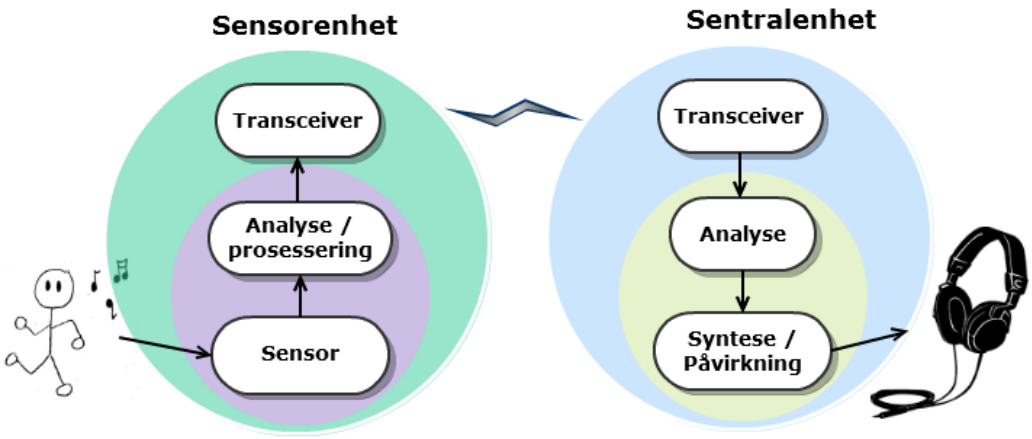


Figur 1.1: Oversikt over et generelt Aktiv Musikk-system.

ninger gir ulike fordeler og ulemper. For å finne en best mulig sammensetning må man ta hensyn til bruksområdet til systemet. Det vil si at man må vurdere batterikapasitet, hastighet, stabilitet, brukervennlighet, tilkoblingsmuligheter og hvilke sensorer man ønsker å benytte. Avsnitt 2.8 tar for sen en vurdering av slike plattformer.

Figur 1.1 viser hvordan et generelt Aktiv Musikk-system er satt sammen. Bevegelse er grunnlaget for hele systemet. Denne bevegelsen blir oppfattet av en sensor (persepsjon) og digitalisert. Deretter blir det gjort en analyse på sensordataene for at systemet skal forstå hva slags bevegelse som er gjort. Denne analysen kan gjør ved hjelp av maskinlæring som er beskrevet i avsnitt 2.3 eller andre metoder. Videre må man ha en forståelse av hva slags sammenheng denne bevegelsen har til lyd og musikk. Slike sammenhenger er beskrevet nærmere i avsnitt 2.1. Til slutt blir denne sammenhengen brukt til å påvirke musikken som spilles, eller til å syntetisere ny musikk.

Figur 1.2 viser et *trådløst* Aktiv Musikk-system og består av en *sentralenhet* og en eller flere *sensorenheter*. Som navnene tilsier er det sensorenheten(e) som foretar innsamling av sensordata, og gjør en eventuell analyse av dataene før de sendes videre til sentralenheten. Sentralenheten tar i mot data fra sensorenheten(e), gjør eventuell videre analyse og syntetiserer/påvirker musikken. En eller flere sensorenheter kan også kommunisere med andre sensorenheter, hvis dette er hensiktsmessig for systemet.



Figur 1.2: Sentralenhet og sensorenhet i et trådløst Aktiv Musikk-system. [23]

Hva slags maskin- og programvare disse enhetene skal bestå av, avhenger som nevnt av hvilke egenskaper man ønsker at systemet skal ha. Enkelte systemer har behov for høy overføringshastighet, mens effektforbruk ikke er kritisk.

Andre systemer er avhengig av et lavt effektforbruk, mens høy overføringshastighet ikke er nødvendig. Slike faktorer må vurderes nøye for å finne teknologi og elektronikk som passer best mulig til systemet man ønsker å lage. Dette gjelder både enheten(e) som samler inn, analyserer og sender sensordataene videre, og også selve sensorene. Det finnes et stort utvalg sensorer som registrerer bevegelse på ulike måter. Hvilke man velger, avhenger blant annet av hva slags bevegelse man ønsker å registrere, effektforbruk, størrelse og hastighet.

1.2 Motivasjon og spørsmål

«Sensing Music-related Actions» (SMA) er et samarbeidsprosjekt mellom Institutt for Musikkvitenskap (IMV) og Institutt for Informatikk (IFI). Prosjektet skal finne koblinger mellom bevegelse og musikk, og bruke dette til å utvikle ny musikkteknologi. De viktigste bidragene fra prosjektet, og også motivasjonen for denne oppgaven inkluderer: [15]

- Økt kunnskap rundt bevegelse-lyd-koblinger og deres betydning for analyse og syntese av media.
- Sensorteknologier og metoder for å lese ut enkeltbevegelser fra en kontinuerlig strøm av kroppsbevegelser.
- Prototyper av bærbare mediespillere som gjør det mulig å oppleve Aktiv Musikk.

De to øverste punktene omhandler bevegelsesanalyse og Aktiv Musikk, mens det tredje tar for seg å bruke dette i en mediespiller. Bevegelsesanalysen kan være

- analyse av en eller flere kroppsdeler som beveges,
- en større bevegelse, som å gå fra et sted til et annet,
- eller analyse av andre typer sensorer som indirekte er påvirket av bevegelse, for eksempel måling av puls eller svetting.

På bakgrunn av denne analysen kan man få en god oppfatning av en persons bevegelse. Dette kan videre brukes i en mediespiller til å skape Aktiv Musikk. For å gjøre dette på en best mulig måte har vi en rekke ønsker:

- Finne sensorteknologier som oppfatter bevegelse på en best mulig måte. Til bruk i en håndholdt mediespiller må sensorene i tillegg være kompakte, lette og ha et lavt effektforbruk.

- Finne plattform(er) og teknologi(er) som fungerer godt som sensor- og sentralenhet.
- Finne ut hvordan man best mulig kan bruke ulike sensordata til å kontrollere musikk.
- Forstå hvordan musikk og lyd er koblet sammen med bevegelse i dagliglivet, og bruke dette i Aktiv Musikk-systemer.

Disse spørsmålene er forholdsvis generelle, og alt kan naturligvis ikke besvares i denne oppgaven. Jeg vil derfor vurdere ulike plattformer og velge en av disse. Deretter vil jeg konsentrere meg om denne plattformen og vurdere hvor godt den fungerer i et slikt system. Jeg vil fokusere på enhetene og kommunikasjonen mellom disse, og ikke gå i dybden på analyse og musikksyntese.

1.3 Utfordringer

En av utfordringene med trådløse Aktiv Musikk-systemer er at man ønsker å kombinere forholdsvis høye trådløse overføringshastigheter og mye dataanalyse, med enheter som helst skal ha lavt strømforbruk og tar liten plass. Her må man gjøre flere vurderinger for å finne en optimal løsning. Vurderingen må gjøres på bakgrunn av hva slags sensorer man ønsker å bruke, hvor mye sensordataene skal prosesseeres og analyseres, hvor rask den trådløse overføringen trenger å være, stabilitet og strømforbruk. I tillegg til dette må man se på brukervennlighet. Dette gjelder ikke bare bruken av ferdige systemer, men også programmering og oppsett av nye prototyper. Slik kan nye prototyper og ideer testes av brukere som ikke har så mye programmerings- og elektronikkerfaring.

1.4 Oppbygging

Kapittel 2 gir en beskrivelse av Aktiv Musikk generelt, i tillegg til en beskrivelse av ulik maskinvare, programvare og annen teknologi som kan brukes. Videre kommer en beskrivelse av andre Aktiv Musikk-systemer. Kapittel 3 beskriver detaljert noen implementeringer av Aktiv Musikk-systemer basert på CUI32 og StickOS. Testing av hvordan disse fungerer i praksis beskrives i kapittel 4, med påfølgende diskusjon i kapittel 5. Til slutt vil jeg i kapittel 6 konkludere med hvor godt denne teknologien egner seg til å lage et slikt system, og hvordan den kan og bør utvikles videre. Oppgaven avsluttes med kapittel 7 som kommer med forslag til videre arbeid med systemet.

Kapittel 2

Bakgrunn og teori

Dette kapittelet starter med å gi en beskrivelse av forhold og koblinger mellom bevegelse og lyd. Dette er viktig i Aktiv Musikk som beskrives videre. Deretter kommer beskrivelse av ulik teknologi, maskin- og programvare som kan brukes i et Aktiv Musikk-system.

Det er allerede gjort en del forskning på bruk av bevegelse for å kontrollere ulike enheter de senere årene. Mange spillkonsoller har for eksempel innebygde sensorer som kan registrere bevegelse. Det betyr at man kan spille ved å bevege kroppen, og ikke bare trykke på knapper. Man ser også et økende antall bevegelsessensorer i mobiltelefoner og annen håndholdt elektronikk. Dette åpner for muligheten til å bygge inn bevegelsesanalyse i nye enheter i framtiden. Dette kan være systemer som kobler sammen bevegelse og musikk. For å kunne gjøre dette er en nødt til å ha tro på at det er en sterk forbindelse mellom våre bevegelser, våre tanker og musikk. Disse forbindelsene blir underbygget i kroppslig musikkognisjon[1] som er bygget på miljøpsykologi. Miljøpsykologien tar utgangspunkt i at mennesket er en del av miljøet rundt seg, og at persepsjon (sanseinntrykk) og handling er to sider av samme sak. James J. Gibsons teori [11] om direkte persepsjon, er en del av denne psykologien. Han mener at vår kognisjon[2] ikke er bare noe som skjer i tankene våre. Det er et samspill mellom kropp, tanker og miljøet vi befinner oss i. [17] Disse kan sees på som henholdsvis bevegelse, tanker og musikk. Dette kan forklare noe av grunnen til at «vi er i stand til å *høre* lyder vi bare ser, og *se* lyder vi bare hører» [17].

[1] Gi noe abstrakt eller immaterielt en konkret form.

[2] Tenkning og å tilegne seg kunnskap.

2.1 Bevegelse-lyd

Når man ser en bil kjøre, en dør bli åpnet eller et glass falle mot bakken vet man hva slags lyd dette lager. Uavhengig av om glasset ikke har truffet bakken enda eller om bilen er for langt unna til at du kan høre den. Som vi ser av figur 1.1 er sammenhengen mellom bevegelse og lyd en av de viktigste delene i et Aktiv Musikk-system. A. R. Jensenius har definert to slike sammenhenger som kalles *bevegelse-lyd-koblinger* og *bevegelse-lyd-forhold*. [17]

2.1.1 Bevegelse-lyd-koblinger

Bevegelse-lyd-koblinger beskriver sammenheng mellom bevegelse og lyd hvor det er en mekanisk og akustisk kobling mellom bevegelsen og lyden. [17] Disse koblingene er basert på «Motor Theory of Speech Perception», som beskriver hvordan vi kan skjønne hva noen sier kun ved å se på bevegelsene til munnen eller vice versa. [19] Dette gjelder også andre lyder, ikke bare snakking. Dette kan vi merke hvis vi ser noen spille på et instrument uten at lyden er skrudd på. Vi vil ofte være i stand til å forestille oss hvordan musikken høres ut, bare ved å se. På samme måte vet en musiker hvordan han skal spille på sitt instrument for å lage den lyden han ønsker. Dette er et ønske som blir opprettet i hjernen, gjort om til muskelkraft utøvd på instrumentet, som igjen lager lyd. [17] Slike sterke sammenhenger mellom bevegelse og lyd er gitt i psykologiske studier av lydkilde oppfatning [12] og kan forklares av «mirror neurons» i «ventral premotor cortex» av hjernen. [10]

2.1.2 Bevegelse-lyd-forhold

Bevegelse-lyd-forhold er en større gruppe sammenhenger enn bevegelse-lyd-koblinger. De inneholder alt fra naturlige sammenhenger (for eksempel bevegelse-lyd-koblinger) til mer kunstige sammenhenger. Slike kunstige forhold kan for eksempel dannes elektronisk i en datamaskin. De kan i følge Jensenius deles opp i to grupper: [17]

- **Elektroniske enheter:** I denne gruppen blir lyden som oftest laget ved en fysisk mekanisk interaksjon med enheten, som for eksempel en knapp. Slike enheter finnes det mange av, og noen eksempler er datamaskiner, ringeklokker, mobiltelefoner og elektroniske instrumenter. Her blir lyden produsert elektronisk og gjengitt gjennom en høyttaler.

- **Virtuelle realiteter:** I denne gruppen finner vi filmer, tv-programmer, dataprogrammer, dataspill og andre virtuelle omgivelser. Her blir bevegelsen

gjort virtuelt, som noe vi ser for eksempel på en skjerm, og deretter gjengitt gjennom en høyttaler. Vi holder dataspill innenfor denne gruppen selv om det ofte brukes en fysisk kontroller til å styre spillet. Det er allikevel inne i spillet selve bevegelsen skjer, selv om det innimellom kan være kort vei mellom virtuell og reell bevegelse. Eksempler på dette er bruk av joystick til flyspill, og ratt og pedaler til bilspill.

I slike kunstige bevegelse-lyd-forhold er det opp til «skaperen» og avgjøre hvor naturlig/unaturlig, bra/dårlig, direkte/indirekte osv. forholdet er. Dette gjør at *forhold* aldri kan blir like sterke som *koblinger*. Når vi ser en film har vi forventninger til hvordan ulike hendelser (bevegelser) skal høres ut. Allikevel er filmskaperen fri til å legge på hvilke lyd han vil. Slike forhold kan også endre seg med tid. Før hadde for eksempel de fleste telefoner en noenlunde lik ringetone, mens i dag kan man velge hva man vil. Forventningene til hvordan et slikt forhold skal være, kan dermed bli borte med tiden. [17]

2.2 Aktiv musikk

Vanligvis er vi passive lyttere når vi hører på musikk. Vi hører musikken, men har svært begrenset påvirkning på hvordan den skal være. Vi kan kanskje bare endre volum, hvilke sang vi vil høre, og er vi heldige inneholder musikkspilleren en equalizer[3]. Etter at bærbare mediespillere kom på markedet er det blitt vanlig å lytte til musikk mange ulike steder. For eksempel når vi sitter på toget, er på jobb eller når vi trener. Naturlig nok vil vi gjerne høre en annen type musikk når vi skal slappe av på toget, enn når vi trener. Samtidig har vi forskjellige bevegelser i forskjellige situasjoner. Dette gjøre at man ved hjelp av bevegelse-lyd-forhold kan finne egnet musikk til disse situasjonene.

Folk er allerede opptatt av å påvirke musikken de hører på. Det kan vi se av at mange har store musikksamlinger, vi deler spillelister via tjenester som for eksempel «Spotify», synger karaoke og spiller spill som «Guitar Hero». Dette sier noe om at folk er interessert i å påvirke og lage musikk, men at det kanskje er for vanskelig eller tidkrevende å lære seg og spille et instrument. Ideen er derfor å utnytte bevegelsene vi allerede gjør til daglig. [15] Aktiv Musikk prøver å lage en sammenheng mellom disse bevegelsene og hva som kommer ut av høyttalerne, og dermed gi lytteren en mulighet til å påvirke musikken. Dette kan i hovedsak skje på to måter: [23]

- **Syntese,** hvor brukeren lager ny musikk ut i fra bevegelsene han gjør. Dette kan gjøres enten ved bruk av elektroniske lyder som blir laget i bruksøyeblikket (synthesizer). Det kan også brukes en rekke lyder, eller deler av

[3] Kontroller som kontrollerer lydnivået på bestemte frekvensområder opp eller ned.

lyder, som er lagret i mediespilleren. Disse lydene blir avspilt avhengig av bevegelsene brukeren gjør. Siden sensorene er plassert på kroppen kan brukeren lage nye musikkstykker uten bruk av instrumenter, kun ved å bevege seg.

- **Påvirkning** av allerede eksisterende musikk er også en mulighet. Her vil brukeren for eksempel kunne påvirke hastigheten på musikken, ut i fra hvor fort han går. Bevegelsene kan også påvirke hvilket musikkstykke som skal avspilles. Dette kan gjøres ved at man har et bibliotek av musikkstykker, hvor hvert stykke har noen egenskaper. Disse egenskapene kan for eksempel være energisk eller rolig. Disse er igjen koblet opp mot enkeltbevegelser, eller bevegelse over tid. Dermed kan brukerens bevegelse kobles direkte opp mot hvilke musikkstykke som spilles.

Disse to måtene kan selvsagt utvikles i forskjellige retninger. Allikevel har de til felles at lytteren har innvirkning på musikken han lytter til, og det er derfor dette blir kalt Aktiv Musikk. Et slikt system trenger altså ikke være et eget instrument, men kan tilpasse et stykke til brukerens behov. Parametere som kan endres er for eksempel lengde, rytme, tempo og generelt «humør» i stykket. [15] Enheter på rundt 3-5 sekunder som settes sammen til musikkstykker kalles også «hypermusic».

2.3 Maskinlæring

For å kunne forstå hvordan en person beveger seg ut i fra en kontinuerlig strøm av sensordata, er man nødt til å ha en god forståelse av hva disse sensordataene betyr. Det betyr at enheten som tar i mot sensordata må ha gode algoritmer og analysemetoder for å «forstå» den egentlige bevegelsen som blir gjort. Som vi ser av figur 1.1 er maskinlæring en av analysemetodene som kan brukes. Maskinlæring går i hovedsak ut på å lage programmer og algoritmer som automatisk blir bedre etter hvert som de kjører, på bakgrunn av datasett som beskriver enhetens oppførsel og respons. Dette er nyttig hvis man har store komplekse datasett hvor sammenhenger og mønstre kan finnes automatisk, eller om man har et system som kontinuerlig bør utvikles og forandres etter omgivelsene. Maskinlæring er en form for kunstig intelligens, og metodene som brukes henter kunnskap og erfaring fra blant annet informatikk, biologi, sannsynlighet og statistikk, psykologi og kognitiv forskning. Andre eksempler hvor maskinlæring kan brukes er analyse av medisinsk behandling, ansiktsgjenkjenning eller styring av roboter og maskiner i et skiftende miljø. [21]

2.3.1 Mønstergjenkjenning

Mønstergjennkjenning er en form for maskinlæring. Det brukes for å kategorisere, gruppere og klassifisere mønstre. Et mønster kan for eksempel være en strekkode, et ansikt, eller en bestemt bevegelse som kan påvirke musikk. Mennesker gjør slik gruppering hver dag uten å tenke over det. Eksempler på dette er når vi kjenner igjen mennesker, snakker og leser. Vi ønsker gjerne at maskiner skal kunne gjøre det samme. På denne måten kan man gjøre sorteringen i en fabrikk maskinell, eller man kan få en datamaskin til å skrive en tekst, bare ved å diktere teksten. I et mønstergjenkjenningssystem har vi ofte fem steg mellom mønsteret som samles inn, og kategoriseringen som gjøres. [7]

1. **Sensor** som omgjør bevegelse, bilde, lyd, lukt eller noe annet fysisk til digitale data.

2. **Oppdeling** for å sørge for at objektet eller bevegelsen vi skal se på blir adskilt fra bakgrunnen og andre objekter.

3. **Trekke ut egenskaper** ved objektene eller bevegelsen som kan brukes i klassifisering.

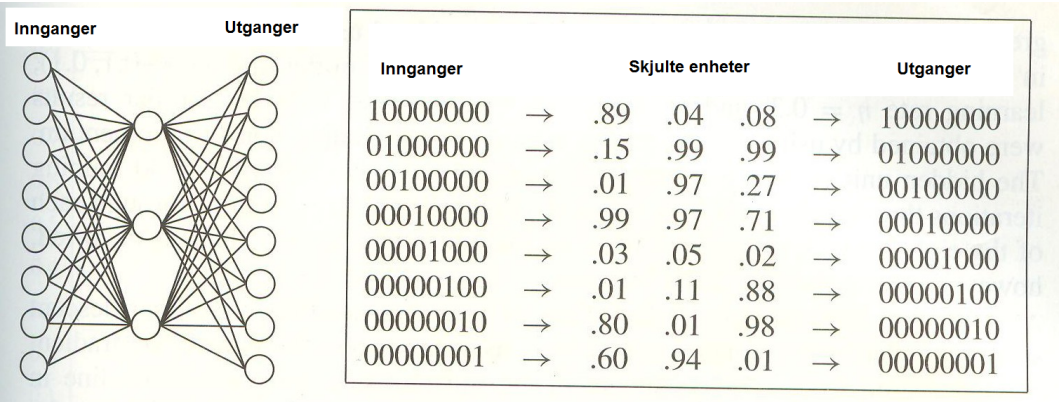
4. **Klassifisering** bruker egenskapene som er trukket ut til å putte objektet i en kategori eller bestemme hva slags bevegelse som er gjort.

5. **Etter-prosessering** brukes for å vurdere om en annen kategori bør velges. Dette kan for eksempel gjøres etter vurdering av kostnader ved feil valg. Dette er aktuelt i et Aktiv Musikk-system hvis det for eksempel er mer sannsynlig at en bestemt bevegelse kommer etter en annen eller lignende.

2.3.2 Kunstige nevronettverk

Et kunstig nevronettverk er et forsøk på å bruke strukturer og funksjoner fra biologiske nevronettverk i maskinlæring. Et slikt nevronettverk består av kunstige neuroner som kobles sammen. På motsatte sider av et slikt nettverk finner vi innganger og utganger. Her er ofte inngangene sensordata, mens utgangene for eksempel kan være motorer som skal styres. Et eksempel på dette er den automatiserte bilen ALVINN, som bruker data fra et 30×32 piksels kamera til å kontrollere styringen av en bil. Her brukes det et kunstig nevronettverk mellom kameraet og styringen av bilen. I et Aktiv Musikk-system vil inngangene være data fra bevegelsessensorer, mens utgangen vil være hvilke bevegelse som er gjort eller hvordan musikken skal påvirkes videre.

Nevronettverk er bygget opp av såkalte perceptroner. De bruker vekter w til å vurdere vektoren på inngangen. Hver vekt w_i multipliseres med tilhørende vektorkoordinat x_i. Ut i fra svaret til denne beregningen kan nettverket ta en avgjørelse. w_i må velges eller trenes opp i læringsprosessen. En algoritme kalt «Backpropagation» sørger for at vi kan ha flere enheter inne i nettverket, i tillegg til innganger og utganger. Disse enhetene er skjult for omverden, og de kan derfor brukes til å automatisk finne nye representasjoner. Figur 2.1 viser et eksempel på dette, hvor et nevronettverk skal lære identitetsfunksjonen. Det viser seg at de tre skjulte enhetene mellom inngang og utgang, automatisk finner standard binær representasjon av tallene fra en til åtte.



Innganger		Skjulte enheter				Utganger
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Figur 2.1: Nevronettverk som skal lære identitetsfunksjonen. Hvis man avrunder de tre skjulte enhetene til nærmeste hele tall, ser vi at vi får binær representasjon av tallene fra en til åtte.

Oppgaver som er spesielt godt egnet til å bruke nevronettverk er oppgaver hvor inngangsdataene til systemet er komplekse og har mye støy. Det er ofte tilfeller for bevegelsessensorer, lyd og video. Som nevnt kan man finne egenskaper som ikke sees direkte fra inngangene og utgangene til systemet. Dette er en stor fordel, siden det gir systemet mulighet til å finne nye funksjoner, uten at brukeren introduserer de.

Det kan være at problem at tilpasningen til treningsdataene blir for god. Da kan vi ende opp med dårligere gjenkjenning av nye data. Dette kalles «overfitting». Dette kan stoppes ved hjelp av kryssjekking for å finne et fornuftig punkt og stoppe algoritmen, før den går «for langt» i opplæringen. Videre er nevronettverk raske å bruke når læringen er gjort, men de har ofte en lengre opplæringstid enn andre metoder som for eksempel valgtrelæring. Allikevel avhenger opplæringstiden mye av antall vekter i nettverket, antall treningseksempler som må brukes og andre parametere til algoritmen.

Som nevnt er denne metoden god når inndataene er komplekse. Dette gjelder for eksempel systemer for tekst-, mønster-, ansikts- og talegjenkjenning, kontroll av kjøretøyer på bakgrunn av kameradata og også bevegelsesanalyse på bakgrunn av data fra bevegelsessensorer. Derfor er denne metoden godt egnet til bruk i Aktiv Musikk-systemer. [21]

2.3.3 Evolusjonell beregning

Evolusjonell beregning (evolutionary computation) er en slags kunstig evolusjon. Den bruker Darwins evolusjonsteori som basis. Det vil si at det opprettes en *populasjon* med ulike løsninger på et problem. Det blir beregnet hvor bra de ulike løsningene er. Ut i fra dette blir de beste løsningene gjort til *foreldre* som lager nye løsninger som kalles *avkom*. I tillegg blir det gjort mutasjoner på andre løsninger og avkom. På denne måten går populasjonen videre, og en vil til slutt sitte igjen med bedre løsninger enn det man hadde fra begynnelsen av.

2.4 Aktiv Musikk-system

For å lage et Aktiv Musikk system trenger man:

- sensorer,
- algoritmer for å analysere bevegelse,
- bevegelse-lyd-sammenhenger,
- musikkstykker og/eller lyder,
- musikkspiller med høyttaler(e),
- og kommunikasjon mellom de ulike delene.

Figur 1.1 viser sammenhengen de ulike delene i et generelt Aktiv Musikk-system. Det *trådløse* Aktiv Musikk-system fra figur 1.2 trenger i hovedsak sensorenhet, sentralenhet og høyttalere/hodetelefoner. I en prototypefase vil det også være nødvendig med en datamaskin som kommuniserer med resten av systemet, for å overvåke og samle sensordata for senere analyse. Denne analysen kan eventuelt gjøres sammen med et «motion capture system». Det vil si et kamerabasert system som fanger opp bevegelse. Dette gjøres ved hjelp av punkter strategisk plassert på personen eller objektet man ønsker å analysere. Disse dataene kan sammenlignes med dataene fra Aktiv Musikk-systemet. Dette vil kunne gi en pekepinn på hvor gode sensorene er, og eventuelt om de trenger kalibrering.

En av de største utfordringene ved et slikt system er å finne sammenhengen mellom de kontinuerlige dataene fra sensorene og bevegelsen brukeren faktisk gjør. Man må med andre ord sørge for at systemet «forstår» hvilke data som er viktige, og hvilke som kan filtreres bort. Sentralenheten er nærmere beskrevet i avsnitt 2.6. Sensorenheten er beskrevet i avsnitt 2.7, mens kommunikasjonen mellom disse er beskrevet i avsnitt 2.5.

2.5 Kommunikasjon

Dette avsnittet skal ta for seg kommunikasjon mellom

- sentralenhet og datamaskin,
- sentralenhet og sensorenhet,
- og plattform på sensorenhet og sensor.

Denne kommunikasjonen kan i noen grad foregå på samme måte, og jeg vil derfor lage en mer generell beskrivelse som begynner med OSI modellen og deretter går dypere inn på ulike kommunikasjonsprotokoller.

2.5.1 OSI

OSI står for «Open System Interconnection» og er en abstrakt beskrivelse av lagene i kommunikasjon og datanettverksprotokoller. Modellen består av syv forskjellige lag, fra det fysiske laget i bunn, til applikasjonslaget på topp. Sammenhengen mellom lagene er slik at et lag alltid henter tjenester fra laget under, og leverer tjenester til laget over. Her er en kort beskrivelse av lagene [23] [32]:

1. Det **Fysiske laget** (PHY) er selve elektronikken i nettverket, og definerer fysiske og elektriske spesifikasjoner. Dette laget har som oppgave å sende og motta den rå strømmen av databiter over nettverket. Her blir kabler og kontakter definert slik at den fysiske forbindelsen blir opprettholdt.

2. **Datalink-laget** lager små pakker av den rå datastrømmen fra det fysiske laget. Dette laget sørger for feilfri overføring av datarammer (frames). Dette laget er igjen delt opp i underlag.

    - **MAC («Media Access Control»)** laget sørger for riktig adressering og kanaltilgang. Hver enhet i nettverket har en unik MAC adresse. Denne adressen inkluderes i adressen som lages av de fem lagene over.

- **LLC («Logical Link Control»)** laget ligger mellom MAC laget og nettverkslaget. Dette laget sørger for multipleksing og de-multipleksing av data transportert over MAC laget. Dette lager ber også om at data må sendes på nytt hvis de er feil gjengitt.

3. **Nettverkslaget** har mulighet til å sende datapakker over flere og ulike nettverk, og det er her ip-adressene ligger. På dette laget finner vi rutere som sørger for at data kommer fram over internett, for eksempel.

4. **Transportlaget** sørger for feilsøking og flytkontroll mellom nettverkslaget og sesjonslaget. Dette laget sørger altså for riktig overføring mellom datamaskiner.

5. **Sesjonslaget** sørger for at brukeren kan opprette en forbindelse eller en sesjon. Denne forbindelsen kan være mellom maskinvare og programvareapplikasjoner. Laget sørger for at sesjonen er sikker, og administrerer også dialogen.

6. **Presentasjonslaget** sørger for at data kan presenteres riktig. Her skjer komprimering/dekomprimering og krypering/de-kryptering. I tillegg sørger laget for at to maskiner som bruker forskjellig dataformat kan kommunisere.

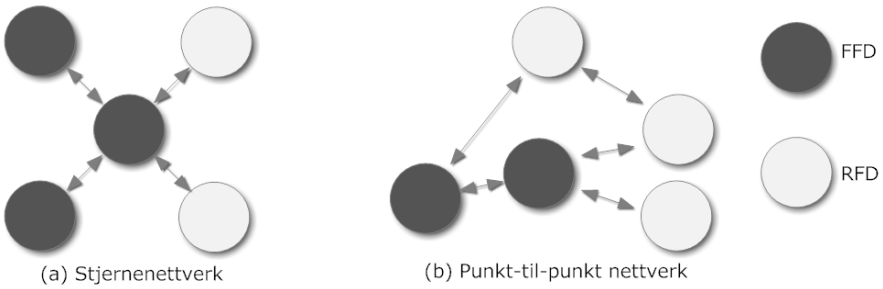
7. **Applikasjonslaget** inneholder selve programvareapplikasjonene. HTTP protokollen arbeider på dette laget og det er her data blir hentet fra, for å kunne presenteres i e-postprogrammer, nettlesere osv.

2.5.2 Topologi

Det finnes ulike topologier, eller måter å koble sammen enheter i et nettverk. De to vanligste er stjernenettverk og punkt-til-punkt nettverk, som kan sees på henholdsvis figur 2.2a og 2.2b. Nettverk består ofte av to ulike enheter, fullt fungerende enhet (FFD) og redusert fungerende enhet (RFD). En RFD brukes til enkle oppgaver og sover mye av tiden. Det kan for eksempel være en sensorenhet som våkner, leser av og sender sensordata, og deretter sovner igjen. En FFD derimot, kan ha større oppgaver og kontrollere hele nettverk. En FFD kan kommunisere med både FFDer og RFDer, mens en RFD kun kan kommunisere med FFDer. [23]

Stjernenettverk

I et stjernenettverk er alle nettverksenhetene koblet sammen til en hovedenhet eller hub, som har ansvaret for kommunikasjonen mellom de andre enhetene. Denne typen nettverk passer godt for et sensorsystem hvor sensorene ikke



Figur 2.2: Stjernenettverk og punkt-til-punkt nettverk. Fullt fungerende enhet (FFD) og redusert fungerende enhet (RFD).

trenger å ha kontakt med hverandre, men kun rapporterer til en hovedenhet. Hovedenheten vil også unngå datakollisjoner og sørge for synkronisering i nettverket. [23]

Punkt-til-punkt nettverk

I et punkt-til-punkt nettverk kan alle enheter kommunisere med alle andre enheter innenfor rekkevidde. Som navnet tilsier er dette kommunikasjon mellom to punkter. Her slipper vi at all kommunikasjon må gå gjennom en enhet, men det kan også gjøre synkronisering vanskeligere. Et stjernenettverk består av mange punkt-til-punkt nettverk. [23]

2.5.3 PAN og WPAN

PAN («Personal Area Network») og WPAN («Wireless Personal Area Network») er nettverk mellom datamaskiner eller andre elektroniske enheter som telefoner og PDAer [4]. Rekkevidden til et slikt nettverk er typisk opp til 10 meter. Kommunikasjonen foregår mellom enhetene i nettverket, men kan også kobles til andre nettverk, for eksempel Internett. Det er kommunikasjon mellom enhetene som er mest aktuelt i et Aktiv Musikk-system. Som navnene tilsier er WPAN den trådløse versjonen av PAN. Eksempler på PAN er USB og firewire, mens eksempler på WPAN er Bluetooth, ZigBee, ZigFlea, IrDA, UWB og Z-Wave. I et WPAN er det meningen at alle enheter skal kunne koble seg til hvilke som helst annen enhet innenfor rekkevidde. IEEE 802.15-standarden spesifiserer syv ulike WPAN kategorier. Jeg vil se nærmere på Bluetooth og ZigBee som bruker henholdsvis IEEE 802.15.1 og IEEE 802.15.4, i tillegg til ZigFlea som er en forenklet versjon av ZigBee. [33]

[4] Personlig Digital Assistent

IEEE 802.15.1 - Bluetooth

IEEE 802.15.1 er standarden som Bluetooth bruker. Den ble i første omgang laget som et alternativ til RS-232 seriell datakabeloverføring. Standarden brukes i alle former for elektroniske enheter som datamaskiner, mobiltelefoner, trådløse høretelefoner, gpser, trådløse tastaturer, skrivere og digitalkameraer. Bluetooth kobles opp i en master-slave struktur hvor en master kan kobles til opp til syv slaver.

IEEE 802.15.4 - ZigBee

IEEE 802.15.4 er en standard som spesifiserer det fysiske (PHY) og MAC laget for lavhastighets WPAN (LR-WPAN). Denne standarden er grunnlaget for blant annet ZigBee som er en fullstendig trådløs nettverksløsning. Det fysiske laget kontrollerer transceiveren som sender og mottar data. Den opererer i hovedsak på $2400MHz - 2483,5MHz$ som gir mulighet for 16 kanaler, men har også mulighet for andre frekvenser (se tabell 2.1) [31].

ZigFlea

ZigFlea er en del av operativsystemet StickOS som er laget for å kjøre på mikrokontrollere. Det er likt ZigBee når det gjelder det fysiske laget, og kan dermed bruke samme type $2,4GHz$ transceivere. Forskjellen ligger i størrelsen på protokollstacken. ZigBee har en protokollstack på minimum $30kb$, mens ZigFlea kun bruker $3kb$. Dette gjør ZigFlea til en lettvekter i forhold til ZigBee. Videre er ZigFlea en punkt-til-punkt protokoll, og har kun mulighet til å sende data i en retning av gangen («half-duplex»).

PHY spesifikasjon: For IEEE 802.15.4 har PHY-laget ansvaret for følgende oppgaver [2]:

- Aktivering og deaktivering av radiotransceiver.
- Detektere signalstyrken som brukes på de forskjellige kanalene (ED).
- Indikere kvalitet og styrke på mottatte pakker (LQI).
- CCA (Clear Channel Assessement) rapporterer om opptatte kanaler.
- Velge kanalfrekvens
- Dataoverføring

MAC spesifikasjon: MAC-laget ligger som sagt mellom de øvre lagene og PHY-laget. Dette laget lager sammenkoblingene mellom de fysiske enhetene, og er i stand til å fungere som fullverdig kommunikasjon hvis kryptering ikke er nødvendig. Laget skal også generere og synkronisere beacon-rammer i tillegg til en del andre oppgaver. [2]:

Sammenligning av Bluetooth, ZigBee og ZigFlea

Som vi ser av tabell 2.1 bruker ZigBee svært lite strøm i forhold til Bluetooth. Av tabell 2.1 ser vi at ZigBee har mye kortere oppstartstid enn Bluetooth. I tillegg er effektforbruket lavere. Bluetooth har raskere overføringshastighet, men til et Aktiv Musikk-system er det også viktig med lavt effektforbruk og kort oppstartstid. Bluetooth bruker stjernenettverk som standard, hvor en enhet er master, mens de andre som kobler seg til er slaver. Disse stjernenettverkene kan kobles sammen til større nettverk. [23] ZigBee kan i tillegg til å kobles opp som stjernenettverk, også kobles opp som mesh nettverk. Dessverre finnes det ikke nok informasjon til å ha med ZigFlea i tabellen. Det kan allikevel sies at den på mange måter er en «lett» versjon av ZigBee. ZigFlea vil bli testet i løpet av denne oppgaven.

	Bluetooth	**ZigBee**
Frekvens	$2,4GHz$	$2,4GHz$, $868MHz$, $915MHz$
Effekt	$100mW$	$30mW$,
Batterilevetid	Dager - måneder	6 måneder - 2 år
Rekkevidde	$10-30m$	$10-75m$
Overføringshastighet	$1-3Mbps$	$25-250Kbps$
Nettverkstyper	Ad hoc, punkt-til-punkt, stjerne	Mesh, ad hoc, stjerne
Sikkerhet	128-bits kryptering	128-bits kryptering
Oppstartstid	$3s$	$15ms$,

Tabell 2.1: Sammenligning av Bluetooth og ZigBee. [18]

2.5.4 Transceiver

Transceiveren skal sørge for trådlløs kommunikasjon mellom sensorenhet og sentralenhet. Transceiveren må støtte den standarden man ønsker å bruke til kommunikasjon. Et eksempel på en transceiver som er vanlig å bruke til ZigBee og ZigFlea er produsert av Freescale og heter MC13201 er beskrevet i neste avsnitt.

Freescale MC13201

Freescale sin MC13201 har støtte for SPI (se avsnitt 2.5.5) og kjører på $2,4GHz$. Den har i tillegg støtte for 16 kanaler og en hastighet på opp til $250kbps$. Den støtter både punkt-til-punkt nettverk og stjernenettverk, som er beskrevet i avsnitt 2.5.2 om nettverkstopologi. Den støtter IEEE 802.15.4 standarden, som er den standarden som ZigBee og ZigFlea bruker.

2.5.5 SPI

Serial Peripheral Interface buss (SPI buss) er en standard for synkron seriell dataoverføring. Det vil si dataoverføring som skjer i takt med en klokke (synkron), og hvor dataene blir overført etter hverandre på en linje (seriell). Den opererer også i «full duplex», noe som vil si at data kan overføres i begge retninger samtidig. SPI settes opp slik at den ene enheten er master, mens den andre er slave. Hver enhet har fire pinner som har litt forskjellige navn, avhengig av om enheten er slave eller master. Figur 2.3 viser sammenkobling av to slike enheter. De fire pinnene kalles: [34].

- SCLK - Seriell klokke som styres av master.
- MOSI/SIMO - Master Output/Slave Input, kommunikasjon fra master til slave.
- MISO/SOMI - Master Input/Slave Output, kommunikasjon fra slave til master.
- SS - Slave Selektor, bestemmer hvilke slave som master vil kommunisere med. Denne er aktiv lav.

Figur 2.3: SPI master og slave med pinnenavn. [34]

2.5.6 UART

UART står for «Universal Asynchronous Receiver/Transmitter» og er maskinvare som omgjør data på parallell form til seriell form (sending av data), og motsatt (mottak av data). Når en byte skal sendes, blir denne delt opp i biter av UART på den ene siden, og deretter satt sammen til en byte igjen på den andre siden. Som «Universal» tilsier kan den operere på forskjellige hastigheter og forskjellige spenningsnivåer. Dette avgjøres ofte av protokollen som brukes i sammenheng med UART. Dette er for eksempel RS-232 og RS-422. De aller fleste mikrokontrollere har i dag enten en eller flere UART porter. En UART port består av to tilkoblinger. En for sending (Rx) og en for mottak (Tx).

2.6 Sentralenhet

Figur 2.4 hva sentralenheten er satt sammen av, og er den delen av systemet hvor dataene fra sensorenheten(e) blir gjort om til musikk. Denne enheten består av to deler. Den første delen tar seg av kommunikasjon med sensorenheten(e). Denne kan være enten kablet eller trådløs, selv om den trådløse varianten er vektlagt i denne oppgaven. Den andre er selve mediespilleren som omgjør signalene fra sensorenheten(e) til musikk i en eller annen form, og deretter spiller musikken.

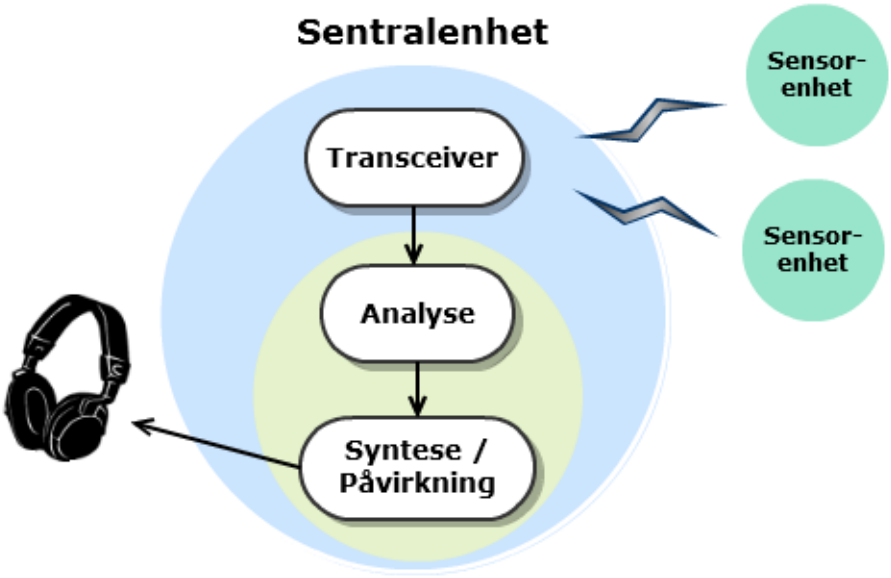
For å få til dette trenger vi en transceiver, altså en sender/mottaker av signalene fra/til sensorenheten. I tillegg trenger vi en plattform (for eksempel mikrokontroller, enkelt-korts datamaskin eller FPGA) som kan ta seg av videre prosessering (se avsnitt 2.8). Dette kan være både beregninger for å komme fram til bestemte bevegelser, filtrering, maskinlæring osv. Til slutt finner vi delen av enheten som syntetiserer eller påvirker musikken.

2.7 Sensorenhet

Denne delen av systemet festes på ulike deler av kroppen og oppfatter selve bevegelsen. Dette gir oss en del begrensninger med tanke på størrelse og vekt. Vi vil helst at brukeren skal bevege seg så fritt som mulig, og ikke hemmes av sensoren. Det finnes en mengde ulike sensorer. Det er for eksempel blitt laget et system som måler puls og elektrisk motstand mellom to steder på kroppen (GSR[5]). På denne måten kan man måle en emosjonell status til brukeren [4].

I tillegg til selve sensoren må man ha en plattform som tar i mot data fra sensoren, og sender disse videre. Avsnitt 2.8 tar for seg en vurdering av ulike

[5] `http://en.wikipedia.org/wiki/Galvanic_skin_response`



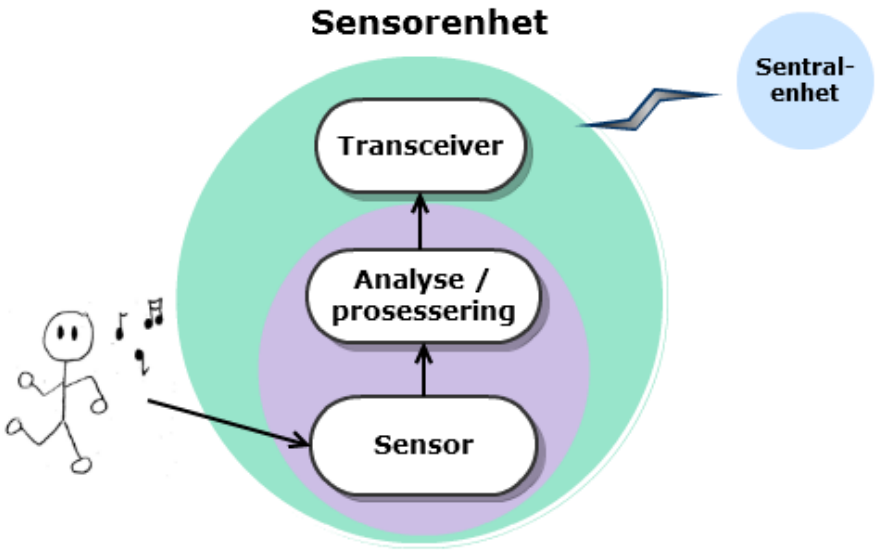
Figur 2.4: Sentralenheten mottar data trådløst ved hjelp av en transceiver fra sensorenheten(e). Plattformen analyseer disse dataene. Deretter blir musikk syntetisert eller påvirket, og til slutt avspilt. I en prototypefase vil som regel noe av analysen og syntesen/påvirkningen bli gjort på en ekstern datamaskin.

plattformer. I tillegg kan det være aktuelt å legge inn dataprosessering på denne enheten, enten for å avlaste sentralenheten eller for unngå å sende mer data enn nødvendig trådløst. Siden kommunikasjonen mellom sentralenheten og sensorenheten skal foregå trådløst er man også nødt til å ha en trådløs transceiver med antenne koblet til plattformen. Figur 2.5 viser hele denne sammenkoblingen. I de neste avsnittene kommer en introduksjon til noen av de vanligste sensorene som kan registrere bevegelse, i tillegg til mer utfyllende informasjon om noen spesifikke sensorer innenfor noen av kategoriene.

2.7.1 Akselerometer

Akselerasjon kommer i følge «Newtons 2. lov»[6] som følge av en kraft. Denne kraften kan for eksempel være at vi trosser tyngdekraften ved å bevege armen opp. Ut i fra akselerasjonen kan vi finne fart og posisjon ved å derivere henholdsvis en og to ganger. Allikevel vil vi kunne få store feil hvis det er mye vibrering eller annen støy i omgivelsene. Derfor bør ikke bevegelsesfrekvensen overskride $1Hz$ hvis et akselerometer skal kunne gi god informasjon om posisjon [8]. I et Aktiv Musikk-system er ikke posisjonen så viktig, da vi ofte kun er

[6] $F = ma$ hvor F er kraft, m er masse og a er akselerasjon



Figur 2.5: Sensorenheten mottar data fra sensorer som registrerer en form for bevegelse. Videre blir disse sensordataene prosessert og analysert i en eller annen grad. Til slutt blir de sendt over til sentralenheten ved hjelp av transceiveren.

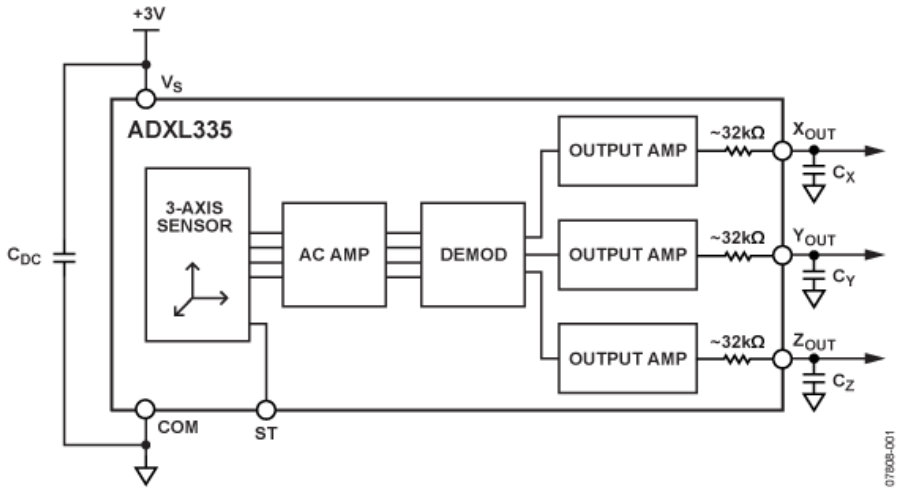
interessert i hvor mye bevegelse det er i et gitt tidsrom. Eller rett og slett om det er bevegelse eller ikke.

Et forholdsvis standard akselerometer er ADXL335 fra Analog Devices som vi ser på figur 2.6. Dette akselerometeret er et 3-akse akselerometer med lavt effektforbruk. Det måler $\pm 3g$ og har tre analoge utganger. [6]

2.7.2 Gyroskop

Et gyroskop er en av de vanligste sensorene innenfor navigasjon, se figur 2.7[7]. Det består som regel av to akser, slik at det kan snus i alle mulige retninger. Når hjulet roterer vil prinsippet om *bevaring av bevegelsesmengde* sørge for at gyroskopet prøver å holde på orienteringen. Et gyroskop som brukes som sensor har en utgang som er proporsjonal med vinkelfarten til hjulet, vinkelrett på rotasjonsaksen. [8]

[7] http://en.wikipedia.org/wiki/Gyroscope



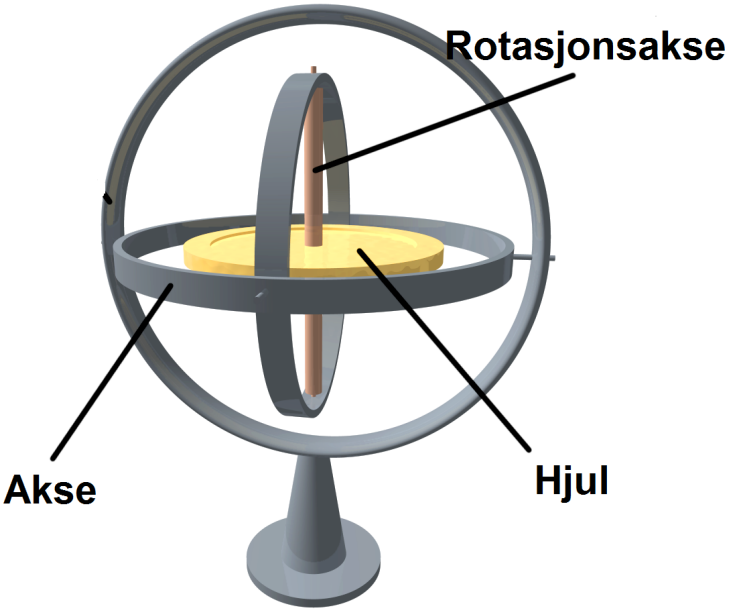
Figur 2.6: Akselerometer ADXL335 [6]

2.7.3 RFID

RFID står for «Radio Frequency Identification». Som navnet tilsier er dette en teknologi som bruker radiobølger til identifisering. En RFID-leser, leser en såkalt RFID-tag. Denne tagen er som regel festet på et objekt, og man kan på denne måten identifisere objektet uten å berøre det. Kommunikasjonen foregår ved hjelp av radiobølger. Det finnes tre typer RFID-tager:

- Passive RFID-tager har ingen egen strømkilde, og trenger derfor et elektromagnetisk felt for å kunne utføre en overføring.
- Aktive RFID-tager har egen strømkilde, og kan sende tagen så snart den merker at en leser er i nærheten.
- Passive RFID-tager med batteriassistanse trenger et eksternt signal for å våkne opp, men har et batteri som brukes til forsterking, slik at rekkevidden blir bedre.

Til et Aktiv Musikk-system vil det være svært virkningsfullt om man for eksempel kan legge sensorenheten over ulike RFID-tager og på denne måten endre lydbildet. Det kan man gjøre fordi hver enkelt tag er unik, og systemet vil derfor forstå hvor brukeren befinner seg. Hvis det er tydelig hvor disse tagene er plassert, vil det være tilstrekkelig med passive RFID-tager. På den andre siden vil det sannsynligvis være mer fornuftig og bruke RFID-tager med egen strømkilde, hvis de ikke er plassert på bestemte oppmerkede steder. [28]



Figur 2.7: Gyroskop

ID-12

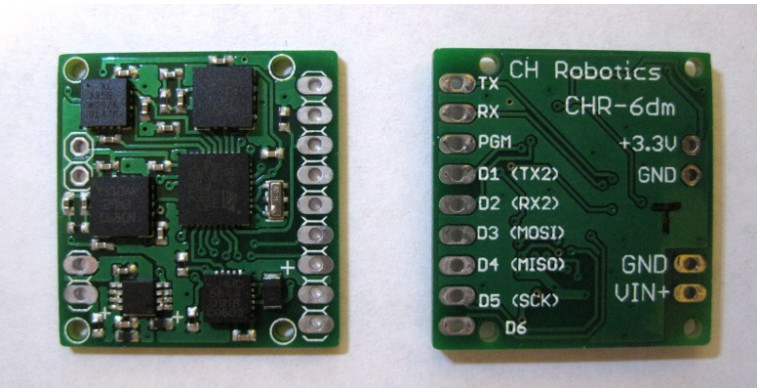
ID-12 er en RFID-leser med integrert antenne på $125kHz$, se figur 2.8. Den har en driftsspenning på $5V$ og kan lese RFID-tager på en avstand på opp til $12cm$. Det betyr at leseren ikke kan plasseres hvor som helst på kroppen. Den må plasseres slik at den kommer nærme nok tagen. Dette kan for eksempel være i skoen til brukeren, hvis tagene plasseres på gulvet. Eventuelt inne i en håndholdt enhet, som kan føres i nærheten av tagene. Videre har den en UART utgang som kan levere både ASCII, Wiegand26 og emulert magnetstripe. [16]

Figur 2.8: RFID-leser, ID-12

Denne sensoren kan som sagt skrive ut både ascii, Wiegand26 og emulert magnetstripe. I et Aktiv Musikk-system vil ascii være det beste. Wiegand26 og emulert magnetstripe kan brukes hvis man har en gammel kortleser som skal byttes ut, uten at man ønsker å gjøre omprogrammering. Man velger hvilke av disse standardene man ønsker ved hjelp av pinne 7, som kalles «Format Selector». Settes denne til jord får man ASCII ut. Datapakkene som blir sendt fra sensoren er formatert som en tekststreng. Den starter med ascii-tegnet STX som er tegnet for start av tekst. [35] Deretter kommer ti tegn som er selve RFID-tagen, og en sjekksum som er den eksklusive OR-funksjonen av disse. Til slutt følger ascii-tegnene CR, LF og ETX. Dette er henholdsvis linjeskift, ny linje og slutt på tekst. [16]

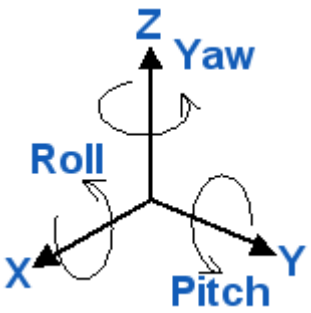
2.7.4 Orienteringssensor CHR-6dm AHRS

CHR-6dm AHRS er en orienteringssensor som inneholder magnetometer, akselerometer og gyroskop, se figur 2.9. Data fra disse blir prosessert gjennom et «Extended Kalman Filter» (EKF), og gir deretter ut yaw, pitch og roll grader direkte. Dette er informasjon som gjør at man vet hvilke orientering sensoren har i tre dimensjoner. Figur 2.10 viser hvordan disse vinklene er gitt. Her er yaw vinkelen vinkelrett på planet til sensoren. Sensoren måler disse vinklene med en nøyaktighet på ned til 2°. I tillegg kan man også få ut rådata fra de ulike sensorene. Disse dataene blir sendt over UART på valgbar frekvens mellom $20Hz-300Hz$. Man kan sette sensoren i «stillemodus», slik at man bare får sensordata når man ber om det. Altså en god løsning hvis man ønsker et system med polling. [24]



Figur 2.9: Orienteringssensoren CHR-6dm AHRS.

Denne sensoren har som nevnt mulighet for både å sende kontinuerlige sensordata og stillemodus, hvor data sendes når de trengs. Pakkene som sendes



Figur 2.10: Definisjon av roll, pitch og yaw grader.

og mottas er definert på en spesiell måte. De begynner begge med en tre-bytes tegnsekvens s,n,p, for å signalisere at en ny pakke kommer. Deretter kommer en byte som kalles PT som spesifiserer hva slags pakke det dreier seg om, og også en byte som oppgir hvor mange databyte som kommer etterpå. Deretter kommer disse databytene. Det kan for eksempel være data fra akselerometeret (sendes) eller hvor ofte sensordata skal sendes (mottas). Til slutt kommer en to-bytes sjekksum. Denne brukes for å kontrollere at dataene som er kommet igjennom er riktige. En vanlig sekvens for å sette sensoren i stillemodus og deretter be om en pakke vil dermed se slik ut, sett fra sensorsiden: [24]

1. Mottar en pakke med $PT = 0x81$ som tilsvarer «SET_SILENT_MODE».

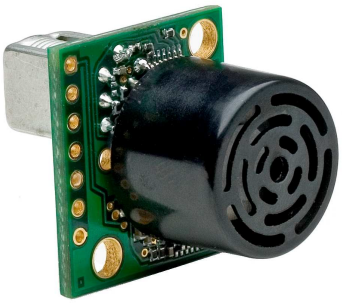
2. Sender en pakke med $PT = 0xB1$ som tilsvarer «COMMAND_COMLPLETE».

3. Mottar en pakke med $PT = 0x01$ som tilsvarer «GET_DATA».

4. Sender en pakke med $PT = 0xB7$ som tilsvarer «SENSOR_DATA». Her sendes da alle sensordataene som sensoren kan levere, med mindre man har brukt $PT = 0x80$ («SET_ACTIVE_CHANNELS») for å bestemme hvilke sensordata som skal sendes.

2.7.5 Avstandssensor MB1210

MB1210 er en avstandssensor (figur 2.11) som bruker sonar for å måle avstander fra $20cm$ til $765cm$, med en oppløsning på $1cm$. Den kan kobles til ved hjelp av UART, og man vil da få avstanden til nærmeste objekt i cm på ascii-form. Man kan også koble til den analoge utgangen og bruke spenningsverdien til å regne ut avstanden. Den kan bruke forsyningsspenning på både $5V$ og $3,3V$.



Figur 2.11: Avstandssensor som bruker sonar, MB1210

2.7.6 Pulssensor

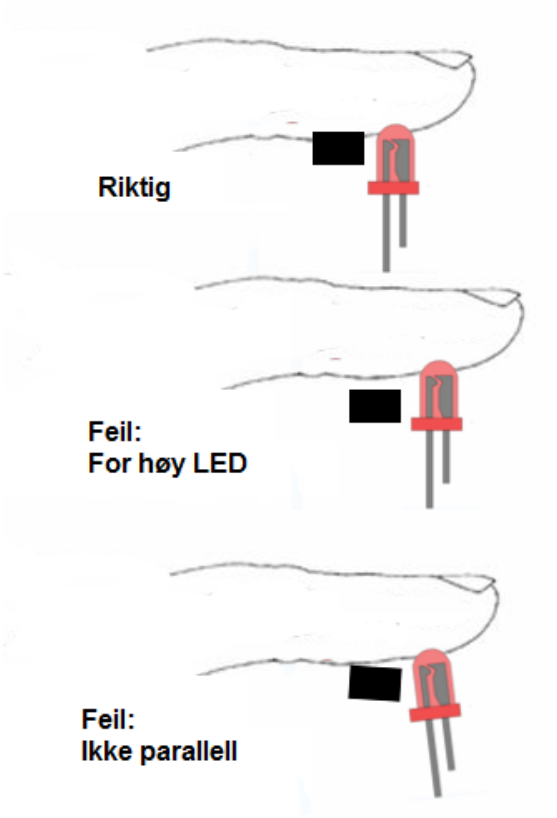
En pulssensor kan lages på flere forskjellige måter. En måte er å bruke en infrarød diode sammen med en infrarød fototransistor. Her vil dioden lyse opp fingertuppen til brukeren. Når blod pumpes gjennom kroppen vil fargen på fingertuppen forandre farge i takt med dette. På denne måten kan fototransistoren «oppdage» når hjertet slår. Dette fordi fargen på fingertuppen vil sørge for at fototransistoren får mer lys, og dermed leder mer strøm. Dette vil gjøre at man får en kurve som endres i takt med pulsen til brukeren. Figur 2.12 viser hvordan fotodioden og fototransistoren bør plasseres i forhold til hverandre, for å gi best mulig resultat.

2.7.7 Sammenligning av sensorer

I dette avsnittet vil jeg gjøre en sammenligning av sensorene som er nevnt i avsnittene over. Tabell 2.2 gir en sammenligning av de ulike typer sensorer. Her er det altså ikke en spesifikk sensorer som blir valgt ut, men for eksempel akselerometere generelt. Jeg vil sammenligne ut i fra hva som er viktig i et håndholdt Aktiv Musikk-system. Dette er hva sensoren måler, altså måleenheten til sensoren. Jeg vil se på den minste bevegelsen sensoren kan måle, om sensoren kan måle sin egen orienteringen i rommet, og hvor mye analyse som er nødvendig før man kan bruke sensordataene til noe rimelig fornuftig. Hva som er rimelig fornuftig er relativt. Allikevel vil rådata fra et akselerometer trenge mer analyse, enn for eksempel en RFID tag.

Tabell 2.3 gir en teknisk sammenligning av de spesifikke sensorene som eer nevnt i avsnittene over. Her vil forsyningsspenning, strømforbruk, og om utgangen er analog eller digital, sammenlignes.

Denne sammenligningen viser at det er en stor forskjell i bruksområdene til de forskjellige sensorene. Akselerometer og gyroskop er gode hvis man ønsker å se på hvor mye en bestemt kroppsdel beveges, eller i hvilken retning denne



Figur 2.12: Retningslinjer for hvordan fotodiode og fototransistor skal plasseres i forhold til hverandre på pulssensoren. [14]

kroppsdelen beveges. Allikevel trenger man mye analyse, for å få nøyaktig informasjon om en slik bevegelse. I en del orienteringssensorer (for eksempel CHR-6dm AHRS) er slik analyse innebygget. Hvis man er fornøyd med denne analysen, kan det derfor være en mulighet å bruke en slik, istedenfor sammensetninger av akselerometere og gyroskop. Hvis man ønsker informasjon om større bevegelse, som forflytning fra et sted i rommet til et annet, vil en av de andre sensorene gi bedre informasjon. Med RFID-tager i gulvet, og leser i skoen vil man kunne følge brukeren rundt i rommet. Dette vil ikke kreve analyse, siden man til enhver tid vet nøyaktig hvor brukeren tråkker. Da kan man også bruke avstandssensorer i ulike retninger, for å se hvilke retning brukeren beveger seg, etter å ha tråkket på en tag. Pulsmåleren registrer ikke bevegelse direkte, men om man beveger seg mye vil pulsen gå opp. Den kan derfor sies å være en indirekte måler.

Når det gjelder de spesifikke sensorene ser vi at forsyningsspenningen ligger på enten $3,3V$ eller $5V$. Ulik spenning gir ekstra jobb hvis de skal kobles til

Sensor	**Hva måles**	**Minste bevegelse**	**Måler orientering**	**Nødvendig analyse**
Akselerometer	Akselerasjon $[m/s^2]$	Liten	Ja	Stor
Gyroskop	Rotasjon [grader]	Liten	Ja	Stor
Orienteringssensor	Roll, pitch, yaw grader	Liten	Ja	Middels
RFID	RFID-tager	Stor	Nei	Liten
Avstandssensor	Avstand [m]	Middels	Nei	Middels
Pulssensor	Pulsslag	Liten	Nei	Liten

Tabell 2.2: Sammenligning av sensortyper.

samme Aktiv Musikk-system, men dette er som regel overkommelig. Strømforbruket er høyest for orienteringssensoren og RFID-leseren, mens akselerometeret bruker mindre strøm. Dette kan være noe å tenke på, hvis man er avhengig av god batterikapasitet og ikke trenger all informasjonen en orienteringssensor gir. Videre er både oppstartstid og oppdateringsfrekvens mer enn god nok for de gitte sensorene, til bruk i et normalt Aktiv Musikk-system.

2.7.8 Dataprosessering

Mange av sensorene som er aktuelle for Aktiv Musikk leverer et analogt signal. Før disse dataene kan sendes trådløst må de digitaliseres ved hjelp av en AD-konverter[8]. De fleste mikrokontrollere har dette innebygget. FPGAer derimot har som regel ikke det, og man må bruke en ekstra chip. Dette gjør bruken av FPGA mer tungvint hvis man skal bruke analoge sensorer.

I tillegg til dette kan man gjøre prosessering som begrenser mengden av data som må sendes til sentralenheten. Man kan for eksempel beregne gjennomsnittet av data innenfor en viss tidsperiode, plukke ut de mest interessante data og kun sende disse eller kun gi beskjed om spesielle sensordata kommer. Det viser seg også at det kan være mer hensiktsmessig å gjøre beregninger på sensorenheten om man bruker et stjernenettverk, enn om man bruker et punkt-til-punkt nettverk. [23]

[8] Analog-Digital-konverter

Sensor	**Forsynings-spenning (Vdd)**	**Strøm-forbruk**	**Analog / Digital**	**Oppstart**	**Oppdaterings-frekvens**
ADXL335 Akselera-sjon	$1,8V - 3,6V$	$350\mu A$ $(3V)$	Analog	$1ms$	-
CHR-6dm AHRS Ori-entering	$3,3V - 9V$	$50mA$ $(3,3V)$	Digital (UART)	-	$300Hz$
ID-12 RFID	$4,6V - 5,4V$	$30mA$ $(5V)$	Digital (UART)	-	-
MB1210 Av-stand	$3,3V - 5V$	$2,1mA$ $(3,3V)$	Begge	$175ms$	$10Hz$

Tabell 2.3: Sammenligning av spesifikke sensorer.

2.8 Ulike plattformer

Dette avsnittet tar for seg ulike plattformer eller utgangspunkt for å lage et Aktiv Musikk-system. Disse kan brukes som grunnlag både for sensorenheten og sentralenheten. De som er beskrevet her er Gumstix som er en enkelt-korts datamaskin, to ulike mikrokontrollere og FPGA.

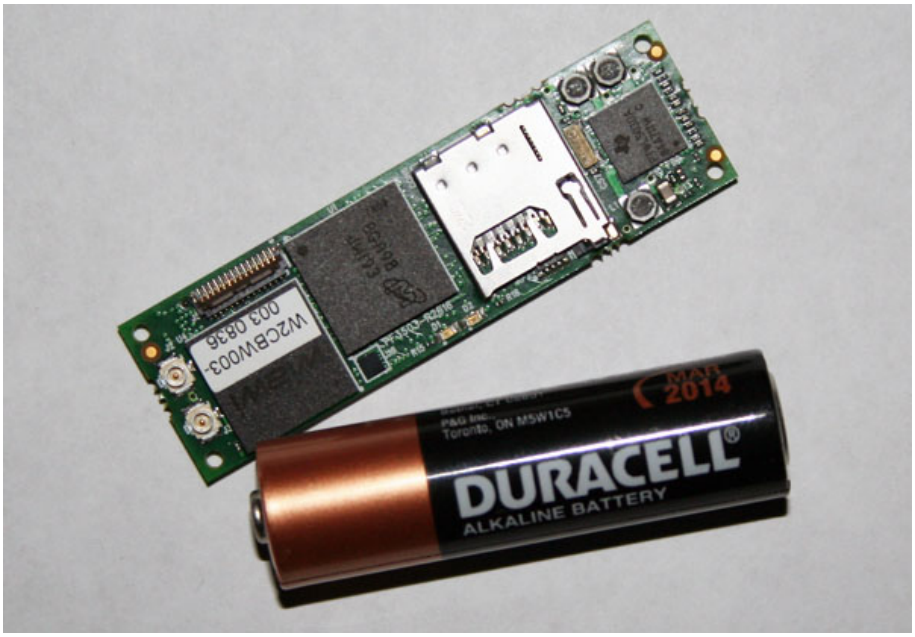
2.8.1 Gumstix

Gumstix var i utgangspunktet navnet på en enkelt-korts datamaskin laget som et hobbyprosjekt. Det har etter hvert utviklet seg til en hel produktserie med slike datamaskiner. Disse maskinene er meget små og et eksempel kan sees på figur 2.13. Det finnes mange forskjellige versjoner, og den mest avanserte har følgende spesifikasjoner:

- Klokkehastighet: $600MHz$
- RAM: $256Mb$
- Flashminne: $256Mb$
- Trådløs kommunikasjon: IEEE 802.11(g) og bluetooth

I tillegg er det tilkoblingsmuligheter for mengder av tilleggskort, microSD kortleser og tilkobling for hodetelefoner og mikrofon. Den støtter også SPI og UART

(se henholdsvis avsnitt 2.5.5 og refsub:uart), som er standarder for dataoverføring. Disse maskinene leveres med en linuxkjerne som kjører OpenEmbedded som er mer beskrevet i neste avsnitt. Tilleggskortene gjør det mulig å koble til USB, ethernet, LCD-skjermer, touchskjermer, HDMI med mer. [13]



Figur 2.13: En gumstix enkelt-korts datamaskin, sammenlignet med et standard AA-batteri. [13]

OpenEmbedded

OpenEmbedded skal gjøre det enkelt å lage embeddede løsninger på linuxkjernen. Embeddede systemer er elektronikksystemer som er spesielt tilpasset en enkelt oppgave, som for eksempel et Aktiv Musikk-system. Typisk for oppgavene til et embedded system er at andre grensesnitt enn skjerm og tastatur brukes, for eksempel bevegelsessensorer. OpenEmbedded har støtte for mange ulike maskinvarearkitekturer og har mulighet for å kompilere en rekke programmeringsspråk. [27]

2.8.2 Mikrokontroller

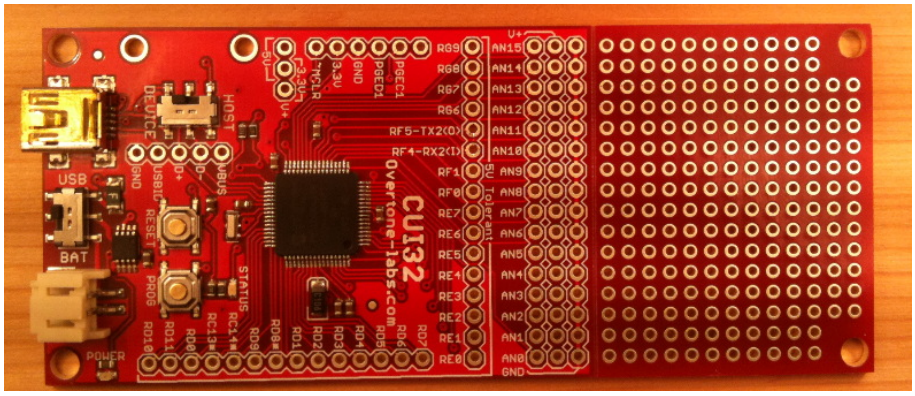
En mikrokontroller er en liten datamaskin på en integrert krets. Den inneholder som regel prosessor, minne og ulike inn- og utganger for tilkoblinger av sensorer eller andre styringssignaler. Den kan programmeres etter eget ønsker

og egner seg derfor godt til spesielt tilpassede elektronikksystemer, slik Aktiv Musikk-systemer er et godt eksempel på. En mikrokontroller har også et avbruddssystem som gjør det mulig at eksterne chiper eller signaler kan avbryte prosessering. Dette er viktig hvis man skal få plukket opp data fra sensorer som sjeldent sender data. [9]

CUI32

CUI32 er et mikrokontrollerkort med USB grensesnitt basert på en 32-bit PIC mikrokontroller (PIC32MX440F512H[9]), se figur 2.14. Det leveres med StickOS ferdig installert noe som gjør at den lett kan programmeres ved hjelp av terminalemulator. Det er også mulig å legge inn annen firmware.

Kortet har 16 analoge innganger som gjør det mulig å koble til for eksempel analoge sensorer. Disse inngangene kan også gjøres om til utganger i kortets firmware for å kunne styre for små motorer eller lignende [22]. Mikrokontrollerens to SPI tilkoblinger gjør det enkelt å koble til en transceiver for trådløs kommunikasjon. I tillegg har den 512Kb Flashminne og 32Kb RAM [20]. Dette kortet kan sammen med en transceiver, brukes som sentralenheten i et Aktiv Musikk-system.



Figur 2.14: CUI32 [22]

StickOS er et operativsystem som kan kjøre på en mikrokontroller. Ved hjelp av et terminalvindu kan det ta i mot BASIC kode og kompilere den. I tillegg har det innebygget en BASIC debugger. Dette gjør det lettere å programmere mikrokontrolleren. StickOS har også støtte for trådløs overføring av data, og mulighet for datalogging til USB minnepinne. [5]

[9] http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en535591

Arduino

Arduino er et mikrokontrollerkort med åpen kildekode. Det er designet med tanke på å gjøre elektronikk mer brukervennlig og tilgjengelig for mennesker som har lite erfaring med mikrokontrollere fra før. Kortet består blant annet av en Atmel AVR mikrokontroller, og digitale og analoge tilkoblingsmuligheter. Mikrokontrolleren programmeres i et språk som er basert på «Wiring»[10] og kan minne om C++. Ulike tilleggskort er tilgjengelig, blant annet «XBee» som sørger for at flere Arduino-kort kan kommunisere trådløs, ved hjelp av ZigBee-protokollen. [29]

2.8.3 FPGA

FPGA står for «Field-programmable gate array» og er en integrert krets som kan tilpasses av brukeren. Den inneholder en rekke logiske blokker og minneblokker som kan kobles sammen. Dette gjøres ved hjelp av et programmeringsspråk som ofte kalles VHDL eller HDL («Hardware Description Language»). På denne måten får man satt opp en krets som er tilpasset akkurat den arbeidsoppgaven man ønsker å utføre. Dette vil i praksis si at den arbeider veldig raskt når den er programmert. På den andre siden er den mer omfattende å programmere, enn for eksempel en mikrokontroller. Videre er det et fåtall av FPGAer som har innebyggede ADCer. Det vil gjøre det mer tungvint å bruke analoge sensorer i et Aktiv Musikk-system. [30]

2.9 Andre Aktiv Musikk-systemer

Dette avsnittet presenterer andre Aktiv Musikk-systemer. Jeg vil i kapittel 5 sammenligne disse systemene med systemet jeg valgte å teste.

«Sense/Stage» [3] er et forskningsprosjekt som utvikler en trådløs infrastruktur for sceneopptredener og interaktive sanntidsmiljøer. [3] Prosjektet fokuserer på å lage et billig system som ikke er unødvendig komplisert, men tilpasset artister som driver med Aktiv Musikk eller lignende. På samme tid ønsker de å kunne koble sammen mange sensorer. For å realisere dette brukwe de mikrokontrollerkortet Arduino Mini Pro sammen med transceiverkortet XBee. Arduino er nærmere beskrevet i avsnitt 2.8.2.

Eirik Renton brukte sin masteroppgave på å utforske forskjellige nettverkstopologier med tanke på bruk i Aktiv Musikk-systemer. [23] Brukte en mikrokontroller fra Atmel, og ZigBee som kommunikasjonsprotokoll. Deretter forsøkte han å finne ut hvor mange akselerometere som er hensiktsmessig å bruke i et

[10] Et annet mikrokontrollerkort med åpen kildekode.

slikt system, uten at tidsforsinkelsen blir for stor. Han kom fram til at i et stjernenettverk er det en øvre grense på tre sensorer per mottaker, men at dette kanskje kan økes til det dobbelte med en mer effektiv implementering. Ved å bruke et punkt-til-punkt nettverk, hvor hver sensorenhet henter data fra flere sensorer før de blir sendt til mottakeren, fant han en øvre grense på fem sensorer. Dette kan tyde på at det er mest effektivt å bruke et punkt-til-punkt nettverk, selv om dette betyr flere sammenkoblinger enn stjernenettverket. Det bør også tas i betraktning at disse tallene er basert på akkurat den mikrokontroller og teknologien som var tilgjengelig på dette tidspunktet. Dermed er nok forholdet mellom antall sensorer det viktigste å merke seg.

«Wireless Remote based Music Synthesizer» eller «KiiWii» er et proskekt som ønsker å bruke Nintendo Wii-kontrolleren til å påvirke musikken som spilles fra et MIDI[11] keyboard. [1] Kontrolleren til Nintendo Wii bruker Bluetooth, noe som ble for dyrt for prosjektet. De valgte derfor å bytte til en PIC18 mikrokontroller og en XBee trådløs modul som bruker ZigBee. I tillegg koblet de til akselerometer fra Freescale og trykk-knapper. Knappene velger hva man vil påvirke i musikken, og bevegelse av akselerometeret sørger for selve påvirkningen. Dette gjøres med den ene hånden, samtidig som man spiller på keyboardet med den andre. De konkluderte med at denne type sensor hemmet keyboardspilling, og at en annen type sensor kunne vært mer hensiktsmessig. I tillegg fant de ut at musikere er mer skeptiske til denne type kontrollere, enn ikke-musikere. Dette er også noe av tanken bak Aktiv Musikk, at ikke-musikere skal kunne påvirke og lage musikk.

[11] Står for «Music Instrument Digital Interface», og definerer noter digital.

Kapittel 3

Metoder og implementering

I de tidligere kapitlene har jeg introdusert ulik program- og maskinvare som kan brukes i et trådløst Aktiv Musikk-system. Disse kan kombineres på ulike måter, med ulikt resultat, og danner plattformen i systemet. Avsnitt 3.1 gir en vurdering av plattformene fra avsnitt 2.8. Jeg valgte en av disse til videre testing. Deretter kommer avsnitt 3.2 som beskriver utviklingen av et kretskort med transceiver, for å muliggjøre trådløs kommunikasjon mellom enhetene. Til slutt i dette kapitlet kommer avsnitt 3.3 som beskriver generell implementering av testoppsett, mens avsnitt 3.4 tar for seg beskrivelse av syv konkrete oppsett med ulike sensorer.

3.1 Vurdering av plattformer

I avsnitt 2.8 om ulike plattformer for et Aktiv Musikk-system har jeg beskrevet tre former for maskinvare. Det er enkelt-korts datamaskin, mikrokontroller og FPGA. I tillegg finnes det mange ulike produsenter innenfor disse kategoriene. Alle har fordeler og ulemper som man er nødt til å vurdere opp mot hverandre når man skal avgjøre hvilke plattform som egner seg best. I en slik vurdering er det viktig å ta hensyn til blant annet:

Stabilitet

Et trådløst Aktiv Musikk-system må være så stabilt som mulig. Både når det gjelder avlesning av sensor(er) på sensorenhet, overføring av sensordata til sentralenhet og videre prosessering på sentralenhet. Med dette menes en god oppetid, det vil si tiden systemet er tilgjengelig og kan anvendes av bruker. I tillegg ønsker man ikke at systemet skal slutte og svare eller stoppe kjøringen av programmet uventet.

I tillegg til dette er et Aktiv Musikk-system avhengig av en rask sammenkobling av trådløse enheter. Hvis systemet skal brukes på en scene, vil det

ødelegge mye om man ikke kommer raskt i gang igjen hvis en av enhetene for eksempel skulle komme utenfor rekkevidde.

Overførings- og prosesseringshastighet

Hastighet på både overføring mellom enheter og intern prosessering bør være mest mulig tilpasset det enkelte systemet. Det vil for eksempel være nødvendig med en høyere hastighet for å overføre kontinuerlige bevegelsesdata, enn å overføre en RFID-tag når den blir avlest. Selv om et system ikke tar skade av «for høy» hastighet, vil det som regel føre til høyere effektforbruk enn nødvendig. Se for eksempel forskjellen mellom ZigBee ($30mW$) og Bluetooth ($100mW$) fra tabell 2.1. Med tanke på at trådløse systemer ofte har batteri som strømkilde, vil det være viktig å vurdere hastighet opp mot strømforbruk. Dette for å få en lengst mulig «oppetid», samtidig som at hastigheten er god nok.

Tilkoblingsmuligheter

Ulike sensorer og trådløse løsninger, trenger ulike tilkoblinger på plattformen. Dette kan være UART, SPI, I2C, analoge innganger osv. Det er en forutsetning at plattformen man velger har mulighet til å koble til sensorene og annen maskinvare man har planlagt å bruke i systemet. Her bør man også legge vekt på om kretskortene er designet slik at tilkoblingene er tilpasset andre sensorgrensesnitt som finnes. Det vil være en fordel om man kan bruke ferdige sensor fra for eksempel Phidgets[1] direkte. Dette er også viktig for brukervennligheten.

Brukervennlighet

Når man skal lage prototyper til Aktiv Musikk-systemer er det viktig å få utført og testet det man ønsker, så raskt og effektivt som mulig. Det beste ville vært om man kunne kombinere god brukervennlighet med ubegrensede muligheter, uten og ha noen særlig kunnskap eller erfaring. Dette er naturlig nok ikke mulig. Dette gjør at vi må finne en plattform brukervennlighet og muligheter, tilpasset utviklerens kunnskap og erfaring. Utviklere av Aktiv Musikk-systemer vil gjerne ha mest fokus på musikk og lyd. Derfor vil brukervennlighet kunne vektlegges sterkt i et slik system.

Kommunikasjon

Et Aktiv Musikk-system kan brukes av en bruker alene, eller det kan brukes på en scene som et instrument eller i forbindelse med en kunstutstilling. Man er derfor avhengig av at den trådløse kommunikasjonen er stabil og har en kort sammenkoblingstid om man skulle komme utenfor

[1] http://www.phidgets.com/

rekkevidde. Her er ZigBee og ZigFlea bedre enn Bluetooth (tabell 2.1). I tillegg brukes Bluetooth på mye forbrukerelektronikk som mobiltelefoner, bærbare datamaskiner og mediespillere. Det vil si at mobiltelefonene til tilskuerene under en konsert, kan forstyrre et Aktiv Musikk-system som bruker Bluetooth. [3]

CUI32 med StickOS installert er en forholdsvis ny plattform, som har fokus på brukervennlighet. StickOS er med på å senke terskelen for hvem som kan programmere en mikrokontroller. I tillegg kan den programmeres i C, hvis det skulle vise seg at StickOS ikke strekker til. Videre har den to UART tilkoblinger, to SPI tilkoblinger, 15 analoge innganger, i2c m.m. Dette burde være tilstrekkelig for de fleste Aktiv Musikk-systemer, med mindre man har spesielle behov. Som tidligere nevnt er dette en ganske ny plattform. Det fantes derfor ikke så mye informasjon om stabilitet og overføringshastigheter. Det lover allikevel godt at man bruker ZigFlea, som bygger på ZigBee, og dermed lover en rask sammenkobling av trådløse enheter (tabell 2.1). I tillegg har en 32-bits mikrokontroller (relativt) god kapasitet til å takle en del prosessering.

Hvis man skal bruke FPGA er man nødt til å programmere i språk som VHDL eller lignende. I tillegg må en lære seg bruk av verktøy for simulering, implementering og debugging. Som nevnt ønsker jeg å se på økt brukervennlighet ved prototyping, så dette gjør FPGA uaktuell i dette tilfellet. Allikevel vil en FPGA gi meget god stabilitet og hastighet, i de fleste tilfeller.

Enkelt-korts datamaskiner, som for eksempel Gumstix er raskere å komme i gang med enn FPGA. Den kan for eksempel programmeres i Python, som er kjent for å være et forholdsvis enkelt språk for nybegynnere. Den har også de fleste nødvendige tilkoblinger. En ulempe er at den bruker Bluetooth som standard. Det vil altså si høyere strømforbruk og lengre sammenkoblingstid av enheter, se avsnitt 2.5 om kommunikasjon.

Jeg valgte til slutt å bruke CUI32 med StickOS som plattform, fordi det er interessant å se i hvilken grad brukervennlighet kan kombineres med høy hastighet og god stabilitet. I tillegg viser erfaring fra musikkmiljøet at Bluetooth ikke fungerer optimalt til Aktiv Musikk-systemer. Dette fordi Bluetooth har en relativt lang sammenkoblingstid (tabell 2.1) og har vist seg å være noe ustabilt. På en scene kan tilskuerenes mobiltelefoner med Bluetooth, være forstyrrende for nettverket til Aktiv Musikk-systemet. [3]

3.2 Egenutviklet ZigFlea-kort

Dette avsnittet beskriver ZigFlea-kortet jeg utviklet til CUI32. Det starter med hvorfor jeg utviklet det, og fortsetter med beskrivelse av skjematikk, utlegg og de ulike delene av kortet.

3.2.1 Utgangspunktet

StickOS bruker, som nevnt tidligere, ZigFlea-protokollen for trådløs overføring. CUI32 har ikke en trådløs transceiver eller antenne. Derfor er vi nødt til å koble til dette for å kunne bruke de trådløse egenskapene til StickOS. ZigFlea bruker det samme fysiske laget som ZigBee (avsnitt 2.5.1 om OSI modellen). Det vil si at vi kan bruke en 2.4GHz ZigBee transceiver. Jeg tok deretter utgangspunkt i et allerede eksisterende kort som kalles CPUstick [2]. Dette kortet består av både en mikrokontroller og en transceiver fra Freescale (MC13201). Jeg koblet sammen *transceiverdelen* på dette kortet til CUI32 ved hjelp av SPI for å sjekke at disse kommuniserte og fungerte som forventet. Ved hjelp av testprosedyren som er beskrevet i avsnitt 3.2.5 testet jeg oppsettet, og det fungerte fint. Som vi ser av figur 3.1 er ikke dette spesielt brukervennlig, med ni ledninger mellom kortene, i tillegg til at det tar unødvendig stor plass. Løsningen ble derfor å lage et eget ZigFlea-kort, spesielt tilpasset CUI32. Dette kortet kan plugges direkte på CUI32, og er derfor mer brukervennlig, og tar mindre plass enn løsningen fra figur 3.1. Det vil også være mer stabilt siden man slipper løse ledninger.

Jeg brukte Eagle Light Edition som er et program for designing av mindre kretskort. Her kan man tegne opp både skjematikk og designe utlegg. Denne prosessen er beskrevet nærmere i avsnitt 3.2.2 og sub:utlegg.

3.2.2 Skjematikk

Med utgangspunkt i skjematikken til CPUStick og databladet til transceiveren fra freescale (MC13201) laget jeg skjematikken til kortet. Det består av noen hoveddeler, som er beskrevet videre i dette avsnittet. Figur 3.2 viser et blokkskjema av hvordan disse hoveddelene er koblet sammen.

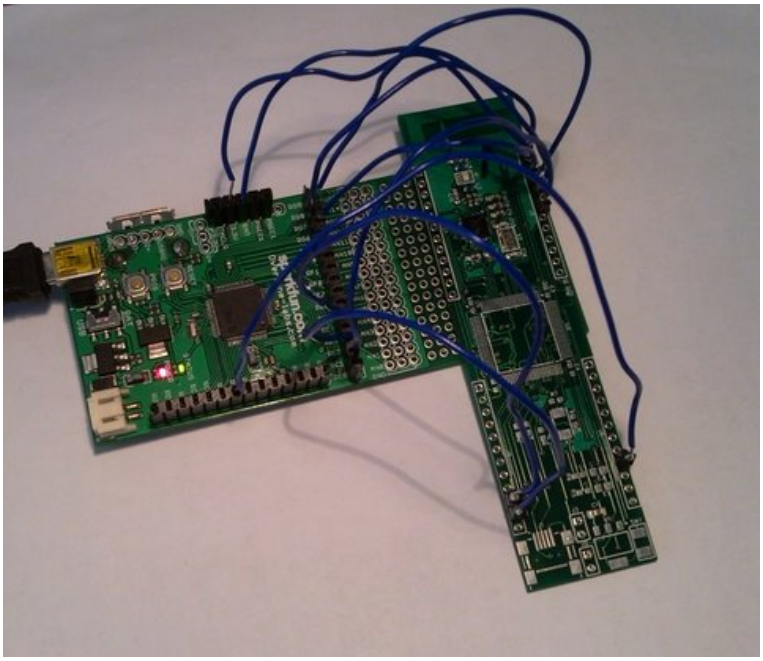
Transceiverchipen (MC13201)

Dette er hjertet på kortet og tar seg av kommunikasjon fra antenne til CUI32. Den er nærmere beskrevet i avsnitt 2.5.4.

Antenne

For å sende data trådløst er man avhengig av en antenne. Den bestemmer i stor grad rekkevidden til systemet, og man må derfor sørge for å designe denne best mulig. Applikasjonsnotatet [25] til transceiverchipen beskriver flere ulike løsninger. Jeg valgte å bruke en F-antenne, som er det samme som er brukt på CPUstick. Denne type antenne er kun synlig på utlegget. Det vil si at antennen er laget kun ved hjelp av kobberbaner på kretskortet og er beskrevet nærmere i avsnitt 3.2.3 om utlegget.

[2] http://www.cpustick.com/cpustick.htm



Figur 3.1: CUI32 til venstre, koblet sammen med transceiverdelen på CPUstick til høyre.

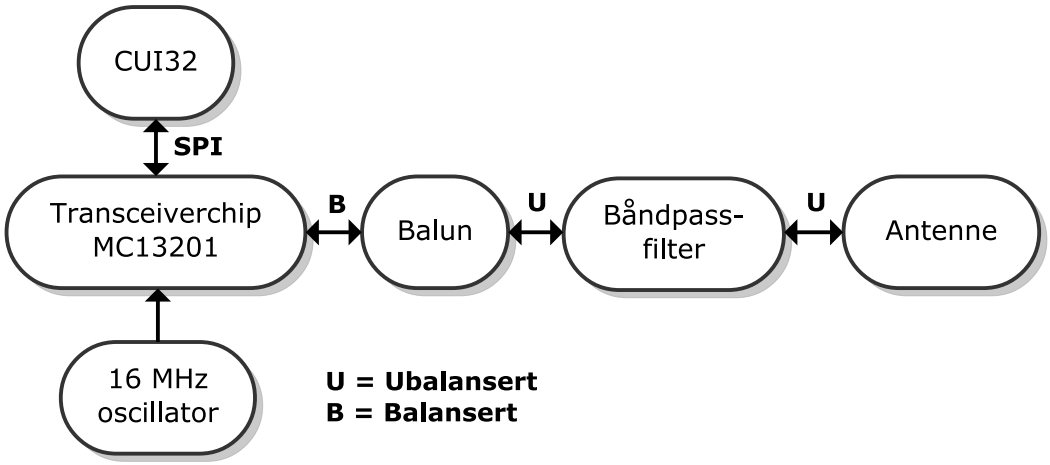
Balun

Signalet som blir mottatt av antennen er ubalansert, det vil si at det ikke er sentrert rundt jord. Ved å bruke en balun og «CT_Bias» signalet fra transceiveren får man et balansert signal som kan sendes til «RFIN_P» og «RFIN_M». Disse to inngangene brukes både til mottak og sending når transceiverens interne bryter for mottak/sending er i bruk. De er henholdsvis positiv og negativ del av det balanserte signalet. «CT_Bias» settes til jord ved mottak, slik at signalet blir «trukket ned» og balanseres til to signaler. På samme måte blir signalet «trukket opp» når «CT_Bias» settes til Vdd ved sending. Oppkoblingen jeg har brukt, og som er anbefalt i databladet kan sees på figur 3.4.

Båndpassfilter

Mellom balunen og antennen er det koblet til en spole og en kondensator som tilsammen utgjør et enkelt båndpassfilter. Dette er for å filtrere bort frekvenser som ligger utenfor båndbredden til transceiveren.

Kortet trenger også en krystalloscillator på $16MHz$ for å styre klokkefrekvensen til transceiveren, i tillegg til en rekke avkoblingskondensatorer på strømtilkoblingene. Dette for å skjerme chipen for raske spenningsendringer som kan



Figur 3.2: Blokkskjema som viser sammenkobling av de viktigste delene på ZigFlea-kortet.

ødelegge den. Figur 3.4 viser den ferdige skjematikken, og er i hovedsak skjematikken som er anbefalt fra databladet til transceiverchipen.

3.2.3 Utlegg

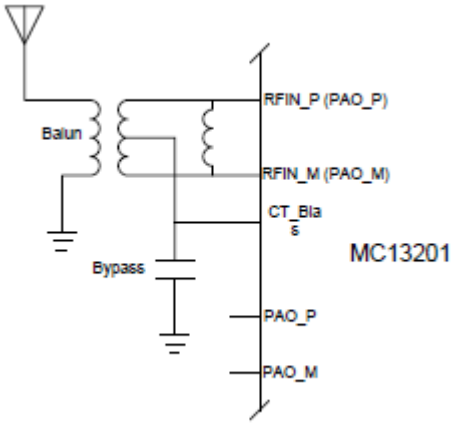
Etter å ha tegnet opp skjematikken i Eagle laget jeg utlegget. Altså hvordan kortet kommer til å se ut fysisk, med kobberbaner og loddepunkter. Under designen la jeg vekt på følgende punkter:

Sammenkobling

Jeg ønsket å lage kortet slik at pinneradene passer nøyaktig over de riktige pinneradene/tilkoblingene på CUI32. Ved å bruke hun-pinner på det ene kortet og han-pinner på det andre kortet, går sammenkoblingen kjapt uten at brukeren trenger å ha oversikt over hvilke signaler som går på hvilke pinner. Dette bygger opp under at Aktiv Musikk-systemer skal kunne brukes av brukere som har liten erfaring med elektronikk og programmering fra tidligere. Figur 3.5 viser de ni sammenkoblingene som trengs mellom kortene og er beskrevet nedenfor:

Vdd
: 3,3V som sørger for forsyningsspenning til ZigFlea-kortet.

Gnd
: Jording til ZigFlea-kortet.



Figur 3.3: Sammenkobling av antenne, balun og transceiver. [26]

IRQ

Forkortelse for «Interrupt Request» og brukes av transceiveren for å gi beskjed til mikrokontrolleren om at det kommer et avbrudd. Denne er aktiv lav, og et avbrudd kan komme for eksempel ved mottak eller sending av data.

RST

Forkortelse for «Reset» og brukes til å nullstille transceiverchipen. Når denne holdes «lav» skrus chipen av og alt som ligger i RAM- og SPI-registre blir slettet. Denne er aktiv lav. Hvis denne holdes «lav» vil transceiverchipen forbli avskrudd, og på denne måten spare strøm.

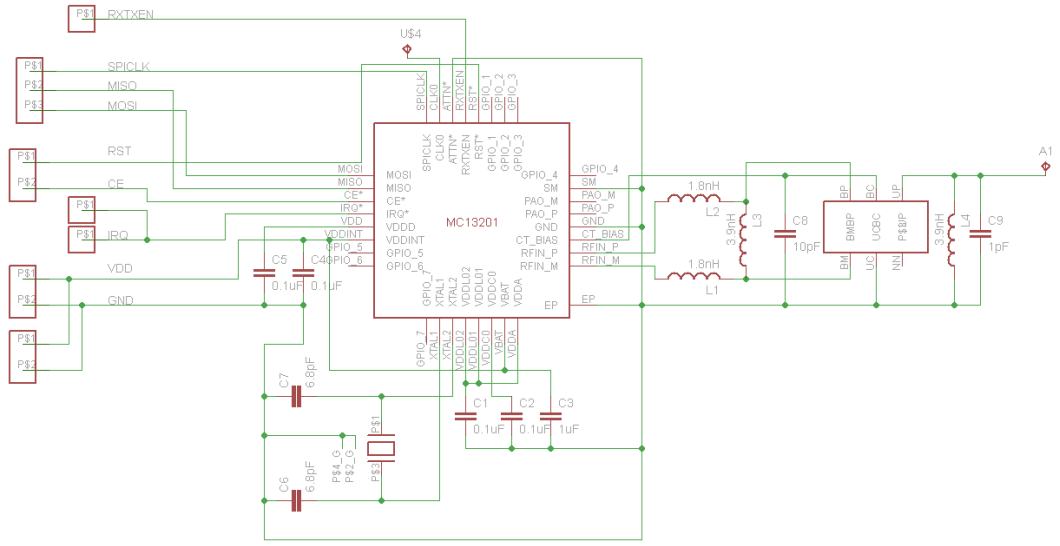
RXTXEN

Skal settes og holdes høy under sending eller mottak av data over SPI. Settes deretter lav igjen.

SPI

Fire standard SPI tilkoblinger for overføring av data. SPI-standarden er nærmere beskrevet i avsnitt 2.5.5.

Som vi ser av figur 3.5 finner vi SPI-tilkoblingene på CUI32 på pinnene RG6, RG7, RG8 og RE1. Vdd og Gnd kobles til 3,3V og Gnd på CUI32, som også kan brukes når kortet programmeres uten å bruke StickOS eller bootloader. De tre resterende kontrollsignalene kobles til pinne RD8 (IRQ), RE2 (RST) og RE4 (RXTXEN). Grunnen til at nettopp disse brukes er at det er slik StickOS er satt opp som standard. Samtidig som at kortet skal dekke de aktuelle pinnene, er det



Figur 3.4: Skjematikk for ZigFlea-kortet.

også nødvendig å gi tilgang til de resterende pinnene som ikke brukes av ZigFlea-kortet. Kortet kan altså ikke være firkantet, men må «gi plass» til pinnene det ikke bruker.

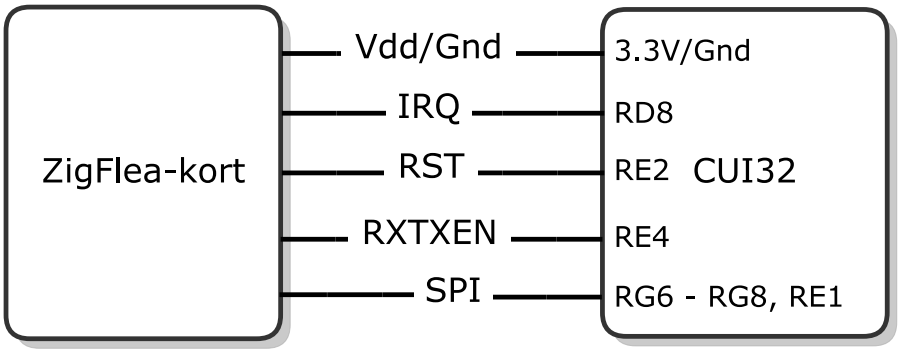
Rekkevidde og antenne

Plassering av antennen på ZigFlea-kortet vil påvirke hvor lang avstanden eller rekkevidden mellom to kort kan være. Jeg prøvde meg derfor fram med transceiverdelen på CPUstick tilkoblet. Jeg testet flere mulige plasseringer. Det viste seg at rekkevidden ble betydelig kortere hvis antennen lå rett over CUI32, enn om den var litt på siden. Dette skjer sannsynligvis fordi CUI32 er dekket av et jordingslag som forstyrrer antennesignalet.

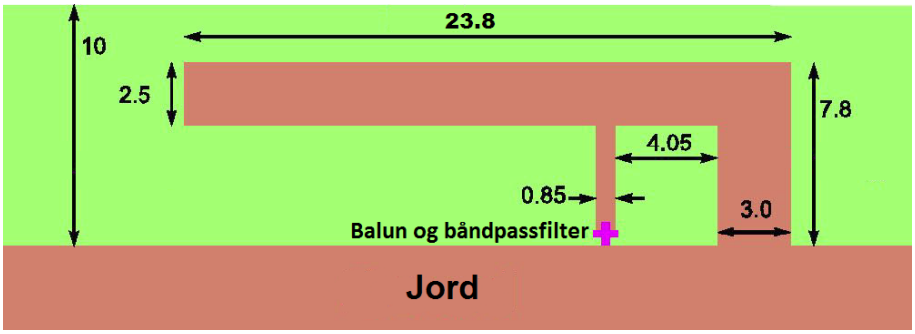
Jeg valgte å bruke samme antenne som er på CPUstick, det vil si en F-antenne. Jeg la den ut ved å ta utgangspunkt i applikasjonsnotatet «Compact Integrated Antenna», som er skrevet av Freescale for denne serien av transceivere [25]. Målene på denne antennen kan sees på figur 3.6. Dette utgangspunktet kan forbedres senere for å øke rekkevidden ytterligere.

Størrelse

Jeg ønsket å lage kortet så lite som mulig, så dette er også prioritert. Allikevel går brukervennlighet og rekkevidde foran størrelsen.



Figur 3.5: Sammenkobling av ZigFlea-kortet og CUI32 med signalnavn, og pinnenavn på CUI32.



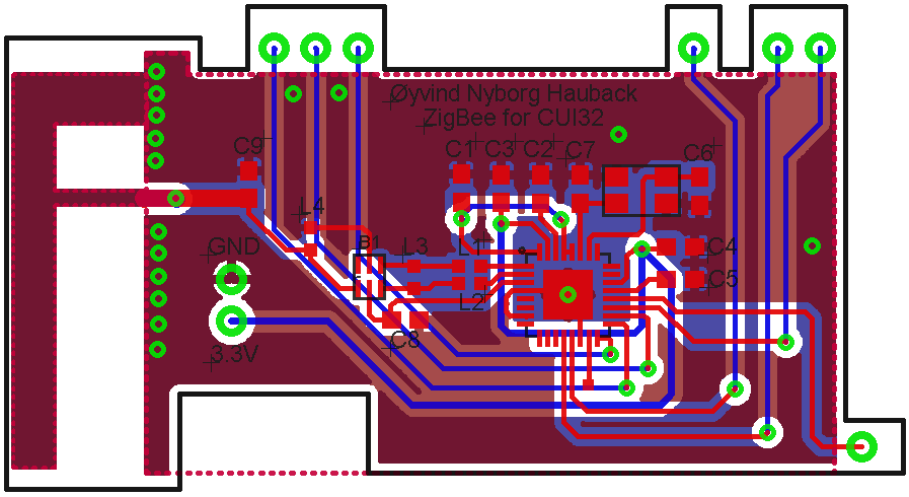
Figur 3.6: F-antenne som er brukt på ZigFlea-kortet. Alle mål er i millimeter. [25]

Konklusjon

Etter å ha vurdert disse punktene opp mot hverandre, innså jeg at kortet måtte bli en del større enn hva antall komponenter skulle tilsi. Det måtte både være stort nok til å dekke de nødvendige pinnene, i tillegg til at antennen må være på yttersiden av CUI32. Dette vil gjøre det vanskeligere å lage en innpakning senere, men rekkevidden blir såpass mye bedre, at dette må prioriteres. Figur 3.7 viser det ferdige utlegget.

3.2.4 Resultatet

På figur 3.8 er det ferdige kortet etter bestilling fra PCB-produsent og lodding. Figur 3.9 viser hvordan kortet monteres oppå CUI32. Her vises de forskjellige tilkoblingene i ytterkant, og antennen til venstre. Ved testing med StickOS fungerer kortet slik det skal. Les mer om hvordan denne testingen er gjort i avsnitt 3.2.5



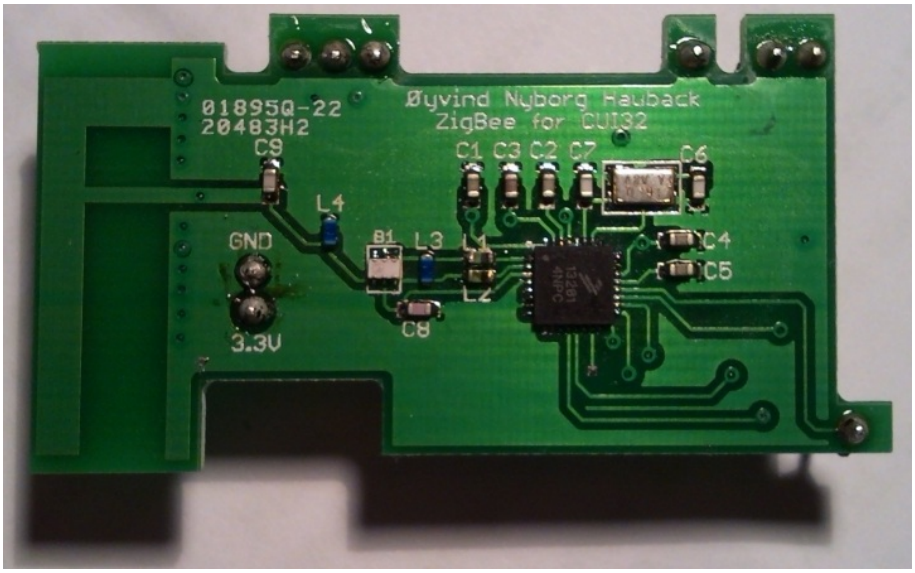
Figur 3.7: Utlegget av ZigFlea-kortet.

3.2.5 Testing

Testing av ZigFlea-kortet gjøres ved å måle spenningsnivået på noen av pinnene til chipen. Hvis disse viser seg å være riktige kan man fortsette testing med med StickOS. Spenningsnivået på pinnene `VDDL01` og `VDDL02` som er koblet sammen, skal være omtrent $1,8V$. Videre skal spenningsnivået på pinne `VBATT` være på omtrent $3,3V$. Om dette stemmer er det ikke blitt kortslutninger eller lignende under lodding. I tillegg kan man finne ut om krystalloscillatoren fungerer som forventet, ved å sjekke at `CLK0` har en frekvens på $32,768Hz$. Hvis dette er i orden fortsetter man med å teste om StickOS og den trådløse funksjonaliteten fungerer som det skal, ved å koble opp slik figur 3.10 viser. Man kan selvfølgelig bruke to terminalemulatorer på *en* datamaskin om man ønsker. Deretter gir man de to navn ved å skrive `nodeid 1` og `nodeid 2` (eller andre tall). Videre skriver man `connect 2` i terminalen til `nodeid 1`. Man skal nå kunne skrive noe, og se at dette endrer seg i terminalvinduet til `nodeid 2`. Hvis dette skjer har man mulighet til å programmere `nodeid 2` i terminalvinduet til `nodeid 1`. I tillegg kan man deklarere fjernstyrte variabler ved å skrive for eksempel `dim variabel as remote on nodeid 2`. Her blir «variabel» navnet på den fjernstyrte variabelen som kan kontrolleres på nodeid 2.

3.3 Implementering av testoppsett

Ved hjelp av ZigFlea-kortet som er beskrevet i avsnittet over, er trådløs kommunikasjon mellom to eller flere CUI32 mulig. Man kobler en strømkilde til CUI32

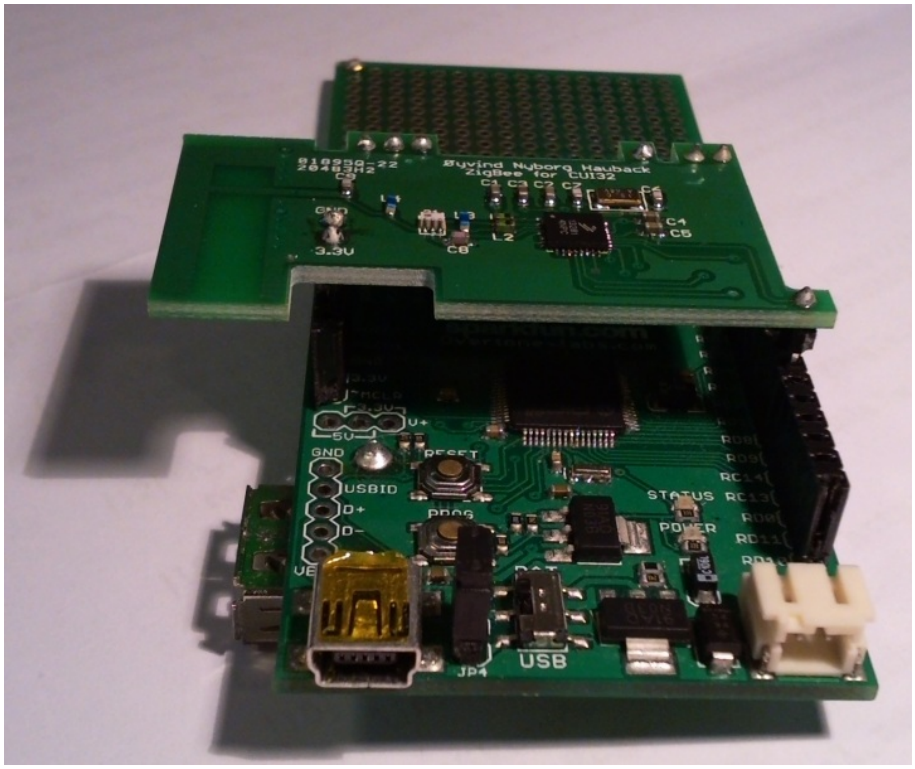


Figur 3.8: Ferdig ZigFlea-kort.

og StickOS starter automatisk opp. Ved hjelp av en terminalemulator kan vi nå kommunisere med kortet og StickOS. Det første vi må gjøre (som nevnt i avsnitt 3.2.5, er å gi de ulike kortene et nr, eller en `nodeid` som StickOS kaller det. Denne bruker vi senere for å deklarere «fjernstyrte» (remote) variabler, altså variabler vi kontroller på et annet kort trådløst. Man kan også koble seg til et annet kort trådløst ved å skrive `connect nodeid`. På denne måten kan vi jobbe på, og programmere et hvilket som helst kort i det trådløse nettverket. Akkurat på samme måte som om det hadde vært fysisk tilkoblet.

Jeg vil teste noen ulike sensoroppsett for å finne ut hvordan disse fungerer sammen med plattformen jeg har valgt. De viktigste punktene å teste er hastighet på trådløs overføring, brukervennlighet, ulike nettverkstopologier og stabilitet. Sensoroppsettene jeg skal teste vil ha ulike egenskaper:

- Oppsett hvor både hastighet på trådløs overføring og hastighet på avlesing av sensorer er viktig.
- Oppsett hvor sensorene har samme grensesnitt/tilkobling mot CUI32 (analogt/digitalt). Dette for å få en renest mulig måling av overføringshastighet
- Oppsett hvor hastighet på trådløs overføring ikke er kritisk, og grensesnittet/tilkoblingene til sensorene er både analoge og digitale.
- Oppsett hvor to sensorenheter sender data til en hovedenhet ved hjelp av ulike nettverkstopologier.

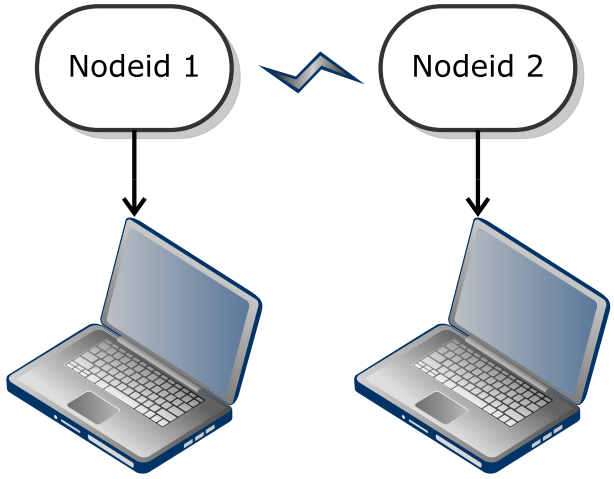


Figur 3.9: Ferdig ZigFlea-kort montert på CUI32.

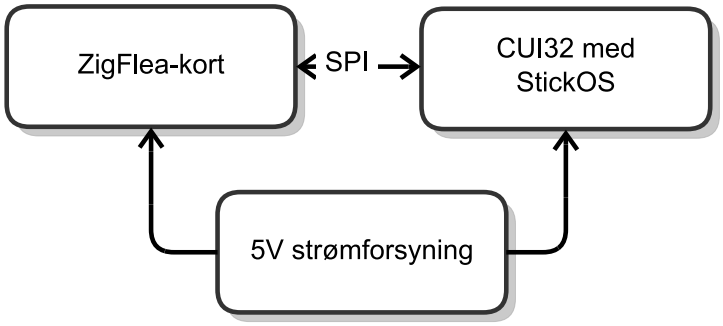
De ulike testoppsettene er beskrevet nærmere i avsnitt 3.4. Her forklarer jeg hvorfor de ulike oppsettene er valgt, og hvordan de er koblet sammen og implementert. Alle sensoroppsettene kobles til samme plattform med strømforsyning, som vist på figur 3.11:

- CUI32 med StickOS installert.
- ZigFlea-kortet for trådløs overføring.
- $5V$ strømforsyning.

Felles for alle oppsettene er at sentralenheten mottar sensordataene og skriver de ut til den virtuelle serieporten. Dette gjør at vi enten kan lese av dataene i en terminalemulator eller et annet program som kan lese fra denne. Hvordan de serielle dataene skrives ut fra CUI32 vises i kodeeksempelet under. Her er `r` en fjernstyrt array som kontrolleres av sensorenheten, og `rfid` er tilsvarende for RFID-tagen. Programmet går i løkke og skriver ut en «A» og deretter sensordata, hvert tiende millisekund. Her brukes en «A» først i hver streng for at programmet som mottar dataene skal kunne kjenne igjen når det kommer en ny rekke



Figur 3.10: Testoppsett for ZigFlea-kortene.

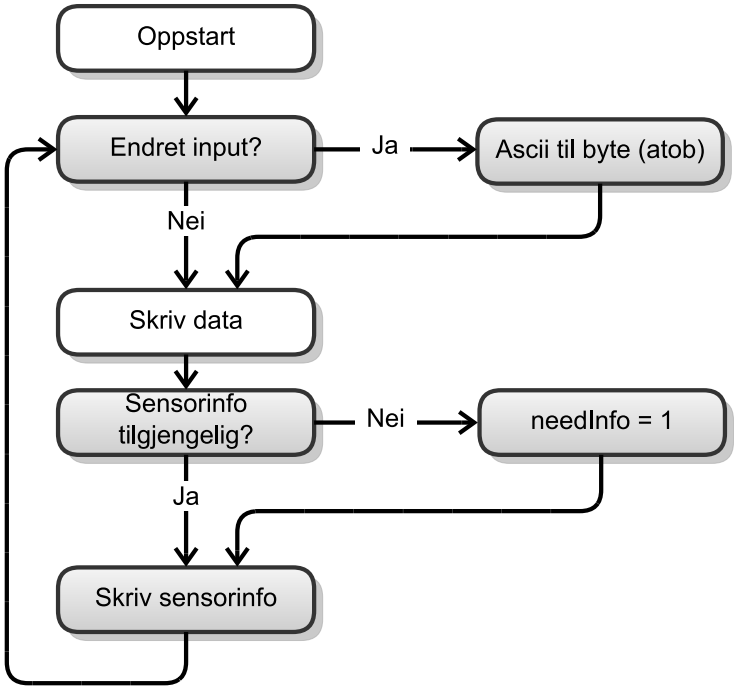


Figur 3.11: Disse tre delene er elles for alle testoppsett. Kalles på senere figurer «CUI32 med ZigFlea-kort».

med sensordata. Dette kan også deles opp mer, for eksempel ved at en RFID-tag starter med «R», mens akselerometerdata starter med «A». Programmet som mottar dataene vil på denne måten kjenne igjen en ny runde med sensordata. Figur 3.12 viser Flytskjema for denne implementeringen. Her er det bare de hvite boksene som gjelder. De grå gjelder for en mer avansert implementering som er beskrevet i neste avsnitt.

```
dim r[16]
dim rfid[10]

while 1 do
        print "A",r,raw rfid
```



Figur 3.12: Flytskjema for kode til sentralenheten. Sentralenheten kan påvirke hvilke sensordata som sendes fra sensorenhet.

```
        sleep 10ms
endwhile
```

Man kan også lage en mer avansert løsning hvor brukeren, ved hjelp av sentralenheten, selv bestemmer hvilke sensordata som skal sendes over. Dette kan gjøres med kodeeksempelet nedenfor. Her bruker sentralenheten kommandoen `input` til å lese en tekststreng serielt, fra for eksempel Max/MSP[3]. Deretter brukes subrutinen `atob` til å gjøre hvert tegn i strengen (som er enten «0» eller «1») om til biter i en byte, hvor 1 og 0 tilsvarer henholdsvis sensor på og av. På denne måten trenger ikke sentralenheten å sende mer enn en byte, for å fortelle sensorenheten hvilke sensordata som ønskes.

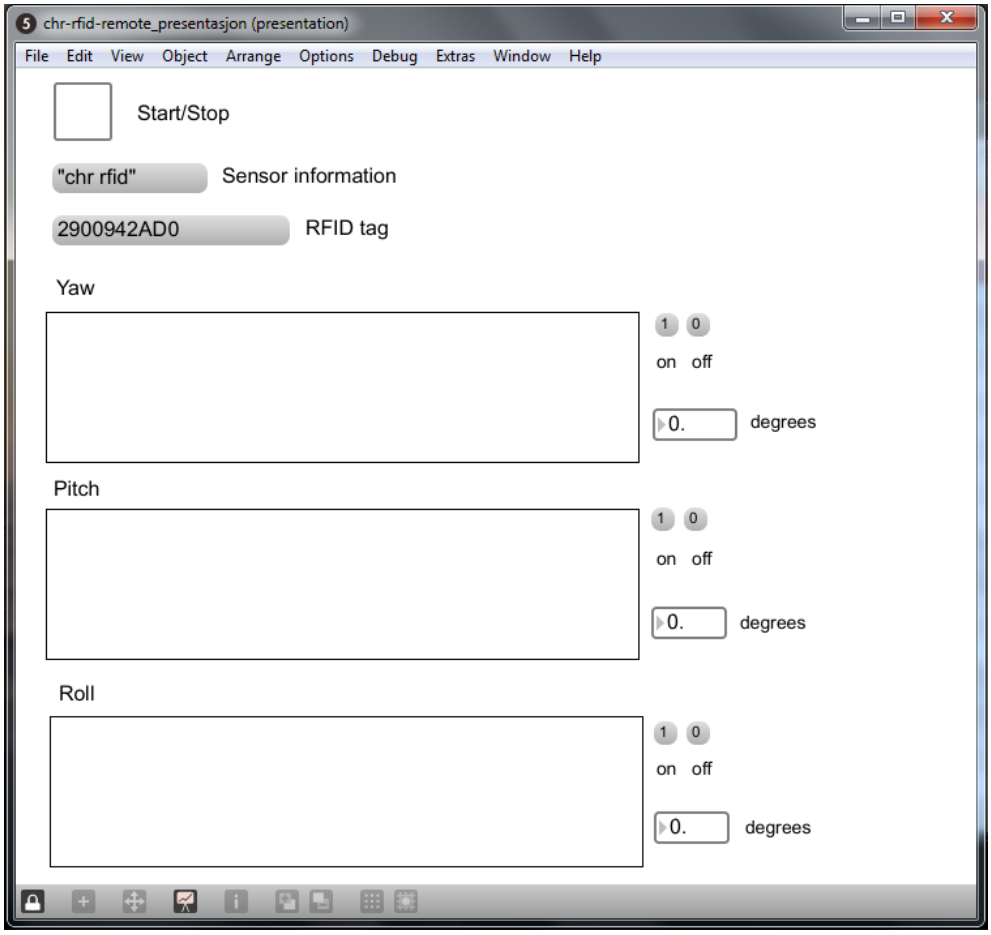
I tillegg mottar hovedenheten info fra sensorenheten om hvilke sensorer som er tilgjengelig. For at sensorenheten skal sende denne infoen, må den fjernstyrte variabelen `needInfo` settes til «1» av hovedenheten. Denne sjekkes av sensorenheten, og info blir om nødvendig sendt. Figur 3.12 viser flytskjema for dette, og kodeimplementering finnes i appendiks B.

[3]Grafisk programmeringsspråk som egner seg godt for lydprogrammering.

3.3.1 Max/MSP patch

I et optimalt Aktiv Musikk-system vil man bruke sentralenheten til å spille musikk, og tilkobling til datamaskin vil derfor ikke være nødvendig. I en prototypefase eller for et system med en sentralenhet som ikke er kraftig nok til dette, vil det være nødvendig å koble til en datamaskin. Terminalemulator er en grei måte å sjekke om et nytt system fungerer, og også sjekke hva slags data som kommer fra sensorene. Derimot egner den seg ikke til å analysere sensordata, og påvirke/syntetisere musikk. Derfor må man bruke et program til å ta i mot sensordataene. I tillegg kan dette programmet også sende data til systemet for å bestemme hvilke sensordata som skal sendes til sentralenheten.

For å lage et slikt program kan man for eksempel bruke Max/MSP som er et grafisk programmeringsspråk. Her får man rask tilgang til serieporten og kan dermed både sende og motta data fra CUI32. Her kan man også gjøre mye prosessering på dataene. I tillegg er det muligheter for å syntetisere lyd direkte i dette programmeringsspråket. Veien fra sensordata til ferdig lyd blir derfor forholdsvis kort. Dermed egner Max/MSP seg godt til å bruke i et Aktiv Musikk-system, så lenge sentralenheten ikke gjør jobben selv. Et eksempel på et program laget i Max/MSP som samler inn sensordata, er vist på figur 3.13.



Figur 3.13: Max/MSP patch som tar i mot informasjon om hvilke sensorer som finnes på sensorenhet. Den mottar sensordata fra sensorenheten, via sentralenhet. Videre kan den bestemme hvilke sensordata som sendes fra sensorenheten, ved å trykke på knappene «1» og «0». I dette eksempelet mottar vi sensordata fra orienteringssensoren fra avsnitt 2.7.4 og RFID-leseren fra avsnitt 2.7.3.

3.4 Beskrivelse av testoppsett

Dette avsnittet tar for seg de syv testoppsettene. For hvert av de beskriver jeg

- hvilke sensorer som testes,
- hva som er spesielt med dette oppsettet,
- og hvorfor jeg ønsker å teste nettopp dette.

Tabell 3.1 og 3.2 gir en rask oversikt over henholdsvis hvilke sensorer som er testet i oppsettene, og hvilke hovedegenskaper som testes i de ulike oppsettene.

Oppsett	1	2	3	4	5	6	7
Orienteringssensor	X						
Akselerometer		X			X	X	
Avstandssensor	X						
RFID	X		X	X			X
Pulssensor			X				

Tabell 3.1: Oversikt over hvilke sensorer som finnes i hvilket testoppsett.

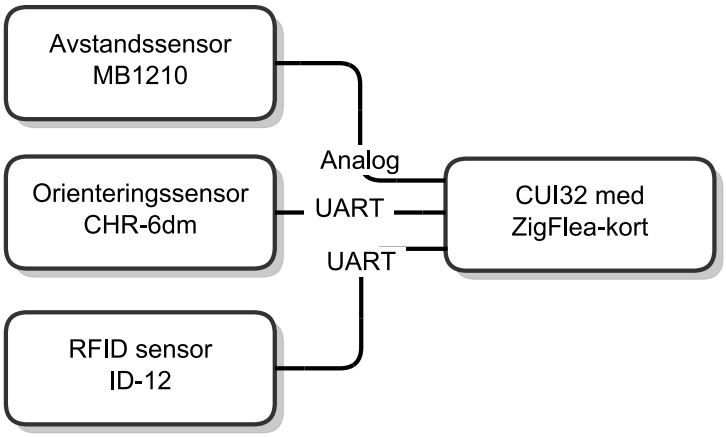
Oppsett	1	2	3	4	5	6	7
Overføringshastighet	X	X					
Stabilitet			X	X			
Nettverkstopologi					X	X	X

Tabell 3.2: Oversikt over hvilke hovedegenskaper som testes i hvilket testoppsett.

3.4.1 Oppsett 1 - Orienteringssensor, avstandssensor og RFID

I dette sensoroppsettet ønsker jeg å teste hvordan plattformen fungerer med tre forskjellige sensorer med ulike grensesnitt:

- Orienteringssensor CHR-6dm fra CHrobotics, med innebygget magnetometer, akselerometer og gyroskop.
- Avstandssensor basert på sonar, XL-MaxSonar-EZ1 (MB1210)
- RFID sensor, ID-12



Figur 3.14: Tilkobling av sensorer for oppsett 1

Den første sensoren (CHR-6dm) er avhengig av at plattformen har en god overføringshastighet for at sensordataene skal kunne utnyttes maksimalt. Dette fordi oppdateringsfrekvensen på kontinuerlige sensordataene bør ligge på mellom $50Hz - 60Hz$ for å kunne gjøre en god bevegelsesanalyse. Denne kan oppdage bevegelse av armer og bein, som skjer hurtig. I tillegg inneholder oppsettet to sensorer som kan oppdage større bevegelser og forflytninger. Disse er ikke så avhengige av høy hastighet. Oppsettet bør derfor kunne oppdage både store og små bevegelser om hastigheten er god nok. I tillegg kobles sensorene til med ulike grensesnitt, både UART og analogt. Disse faktorene gjør at vi virkelig får testet hvor raskt plattformen klarer å jobbe med ulike sensorer og grensesnitt samtidig.

I dette oppsettet er RFID-leseren og orienteringssensoren koblet til ved hjelp av UART, og avstandssensoren er koblet til den analoge inngangen som vi ser av figur 3.14. Avstandssensoren gir konstant ut en analog spenningsverdi for avstanden den måler. Det vil derfor være naturlig og bruke polling for å lese av denne sensoren. Det vil si at man leser av sensoren for en bestemt frekvens.

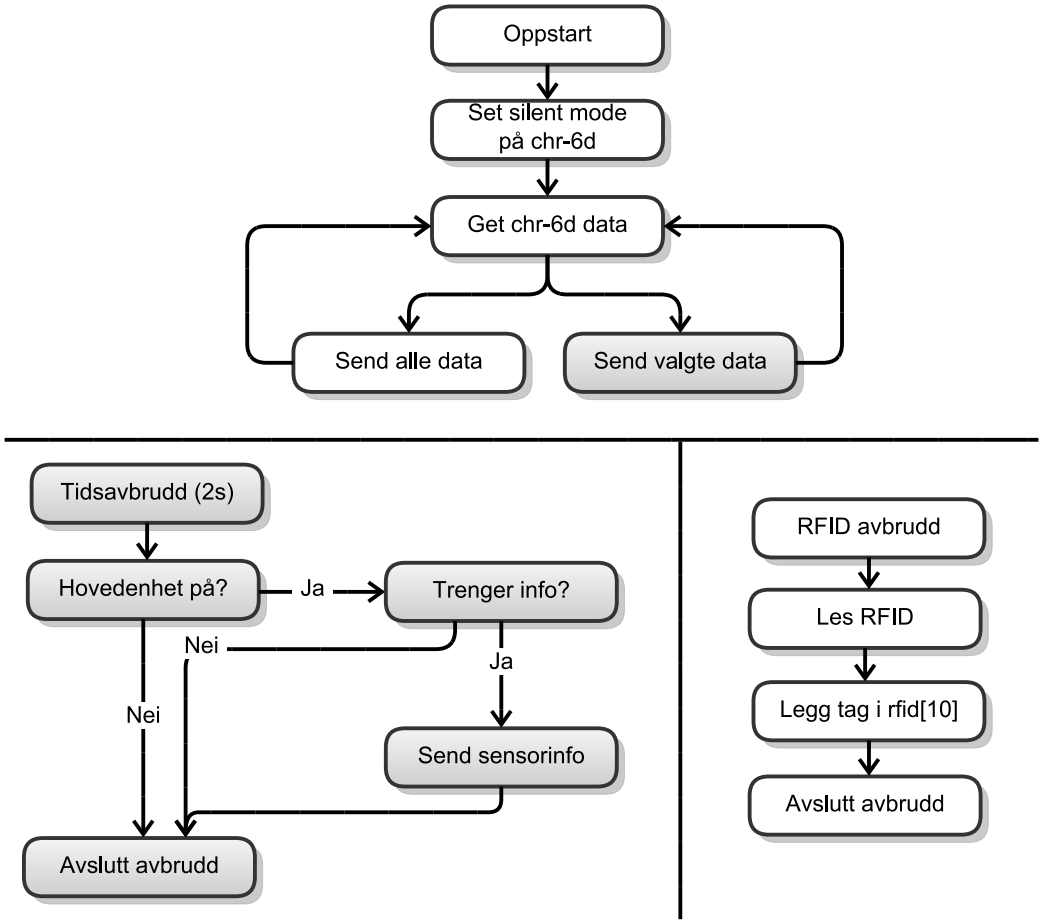
Orienteringssensoren kan sende data på ulike måter. Den kan enten sende rådata fra alle sensorene i tillegg til yaw, pitch og roll grader hele tiden. Den også settes i «stillemodus», og kun sende data når man ber om det. Man kan deretter bruke henholdsvis avbrudd eller polling på CUI32 for å lese ut data. Dette styres ved å sende bestemte beskjeder over UART som beskrevet i avsnitt 2.7.4 om denne sensoren. I tillegg kan man be sensoren om å kun sende dataene vi er interessert i. For eksempel kun data fra akselerometeret. Dette vil begrense UART-trafikken.

RFID-leseren sender kun data når den leser en ny RFID-tag. Det vil derfor være naturlig å bruke avbrudd for å få tak i dataene fra denne sensoren. Hvor-

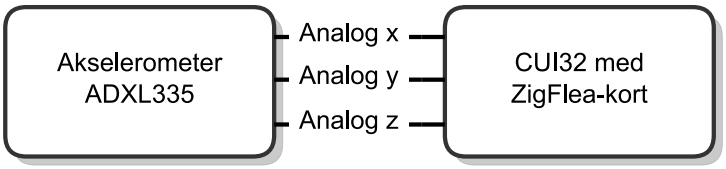
dan man leser data fra denne sensoren er nærmere beskrevet i avsnitt 2.7.3.

Dette oppsettet er som vi skjønner avhengig av to UART porter, hvor den ene bruker avbrudd og den andre polling. Dette for henholdsvis RFID og orienteringssensor. I tillegg brukes en av de analoge inngangene til avstandssensoren. Som sagt får vi derfor testet hvor godt systemet klarer å jobbe med ulike former for sensoravlesing, samtidig som data skal overføres trådløst.

For hver runde med polling, skriver sensorenheten de oppdaterte sensordataene trådløst til sentralenheten ved hjelp av fjernstyrte variabler. Figur 3.15 viser flytskjema for dette. For hver runde skriver programmet ut variabelen `msecs`, som er antall millisekunder programmet har kjørt tilsammen. Dette er en programvariabel som er deklarert og styrt av StickOS. På denne måten kan man se hvor lang tid systemet bruker på å sende en runde med sensordata. Jeg vil på denne måten måle hvor lang tid systemet bruker på å sende alle dataene. Det vil si 15 ulike sensordata fra orienteringssensoren, en fra avstandssensoren og RFID-tager fra RFID-leseren. Deretter senker jeg antallet som sendes, for å se sammenhengen mellom tid og antall sensordata som sendes. Koden i sin helhet finnes i appendiks C.



Figur 3.15: Flytskjema for kode til oppsett 1. Boksene merket med grått er bare med hvis sentralenheten gir beskjed om hvilke sensordata som skal sendes. Under testing er ikke de grå boksene med. Dette for å teste hastigheten på den trådløse overføringen på en mest mulig riktig måte.



Figur 3.16: Tilkobling av sensor for oppsett 2.

3.4.2 Oppsett 2 - Akselerometer

Det andre testoppsettet består av kun en sensor:

- Akselerometer, ADXL335, beskrevet i avsnitt 2.7.1

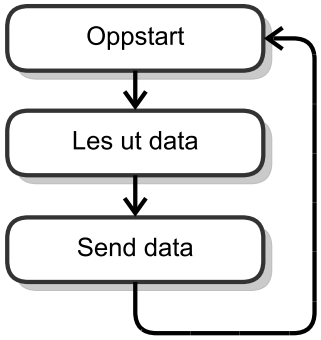
Denne sensoren har tre analoge utganger, som kobles til hver sin analoge inngang på CUI32. Ved hjelp av dette oppsettet får man altså testet hvor raskt CUI32 konverterer fra analogt til digitalt signal (ADC). I tillegg får man testet en forholdsvis «ren» trådløs overføring. Med det mener jeg at det kun er data fra en sensor som skal overføres, uten andre avbrudd eller prosessering. Dette oppsettet vil altså gi en god pekepinn på høyeste oppnåelige trådløse overføringshastighet fra sensorenhet til sentralenhet. Figur 3.16 viser koblingene mellom CUI32 og akselerometeret.

Dette oppsettet bruker på polling på tre av de analoge inngangene for å lese av akseleromteret. En for henholdsvis x-, y- og z-aksen. På denne måten kan man enkelt bestemme hvor ofte data skal avleses og sendes til sentralenheten. På kodeimplementering som er vist i appendiks D er det ikke brukt `sleep` til å legge inn pause mellom hver avlesing/sending. Vi får altså utlest den faktiske tiden som brukes ved hjelp av `msecs`. Denne inneholder som nevnt tidligere kjøretid fra oppstart. Her kunne man gjort en beregning for å finne forskjellen fra forrige `msecs` avlesing direkte. Dette ville på den andre siden tatt mer tid enn å bare skrive den direkte ut. Jeg velger derfor å gjøre beregningen «manuelt» for å få en mest mulig nøyaktig tid. Figur 3.17 viser flytskjema for koden til dette oppsettet.

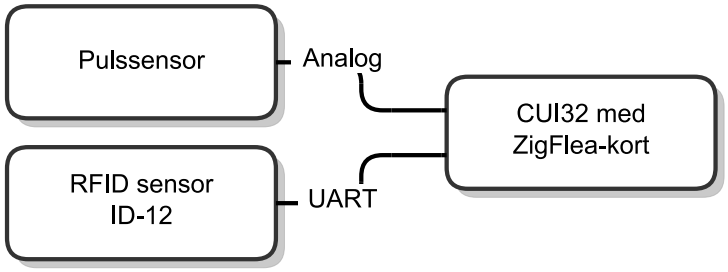
3.4.3 Oppsett 3 - Større forflytninger

Dette sensoroppsettet består av to sensorer som måler større bevegelser/forflytninger:

- RFID-leser, ID-12
- Pulssensor



Figur 3.17: Flytskjema for kode til oppsett 2.

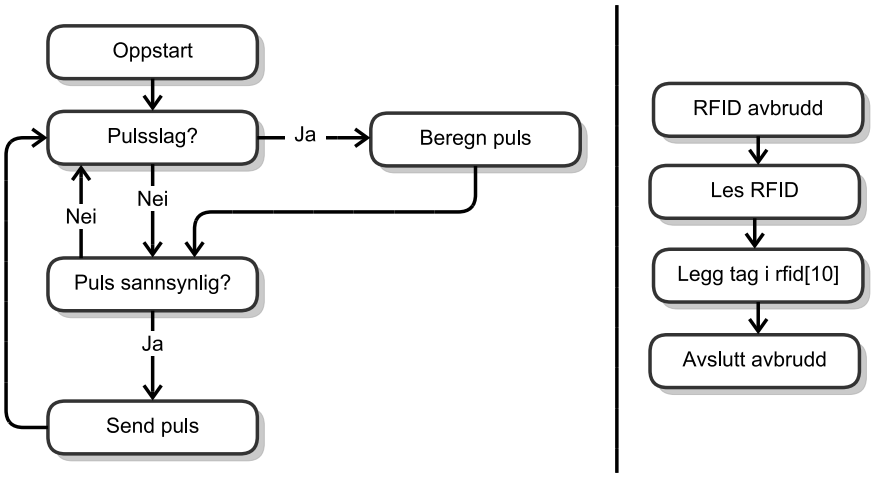


Figur 3.18: Tilkobling av sensorer for oppsett 3

Ved hjelp av dette oppsettet får vi testet hvordan plattformen fungerer med sensorer som ikke er så avhengig av høy trådløs overføringshastighet. RFID-leseren kan registrere hvor man befinner seg i et rom, mens puls kan gi en pekepinn på hvor mye en person har beveget seg den siste tiden. Dette er altså en fint oppsett for å teste hvor godt plattformen egner seg til å registrere større bevegelser. I tillegg brukes både digital (UART) og analog tilkobling, så vi får også testet om dette har noe å si for stabiliteten til systemet. Figur 3.18 viser hvordan dette oppsettet er koblet sammen.

RFID-leseren vil i dette oppsettet leses av når den lager et avbrudd. Pulssensoren kan gi beskjed hver gang pulsen slår, men dette vil kunne bli påvirket av forsinkelser på den trådløse overføringen. For å unngå dette kan man lese av pulsslagene og gjøre beregninger i sensorenheten, og deretter sende over pulsfrekvensen. Dette er mulig ved at StickOS hele tiden oppdaterer variabelen `msecs`, som forteller hvor lenge et program har kjørt. Dette vil ikke bli helt nøyaktig på grunn av at annen prosessering og overføring også tar tid. For å få en mest mulig nøyaktig måling sørger jeg for at pulsen bare blir overført hvert andre sekund. På denne måten rekker vi og få en eller flere pålitelige målinger før målingene blir påvirket av dataoverføringen igjen. Målingene som blir på-

virket vil havne utenfor det som er vanlig puls, altså omtrent utenfor 40 til 200 slag per minutt. Allikevel er ikke nøyaktigheten så kritisk for denne type sensor. Om pulsen måles til for eksempel 62, mens den egentlig er 60 vil sannsynligvis ikke merkes av brukeren. Kodeimplementering av dette kan sees i appendiks E, mens figur 3.19 viser flytskjema for den samme koden.

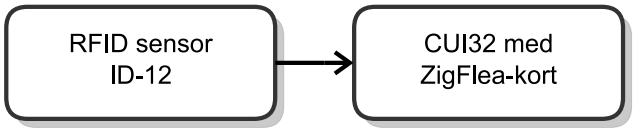


Figur 3.19: Flytskjema for kode til oppsett 3

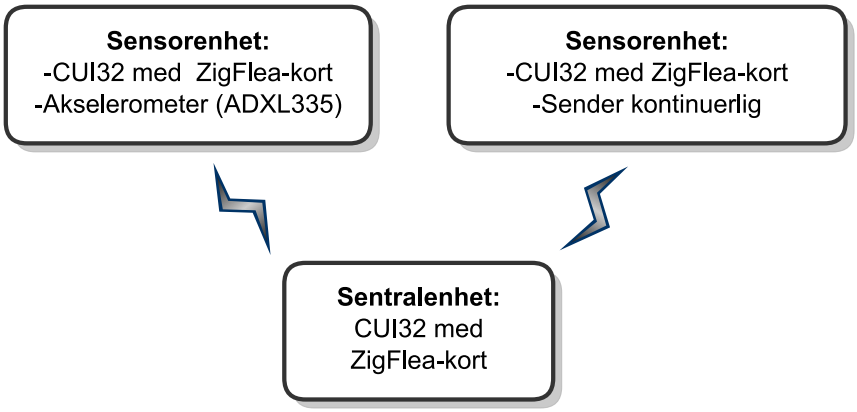
3.4.4 Oppsett 4 - RFID

Dette oppsettet bruker kun av RFID-leser. Det har som hovedoppgave og teste hastighet på avlesing av en RFID-tag, og tiden det tar å overføre hele tagen trådløst. Dette er også et godt oppsett for å teste avbruddssystemet til StickOS separat, uten at andre sensorer «forstyrrer».

RFID-leseren er koblet til ved hjelp av UART, slik vi ser av figur 3.20. Kodeimplementeringen for både sensorenheten og sentralenheten finnes i sin helhet i appendiks F.



Figur 3.20: Tilkobling av sensor for oppsett 4.



Figur 3.21: Tilkobling av sensorer og sammenkobling av sensorenheter og sentralenhet for oppsett 5.

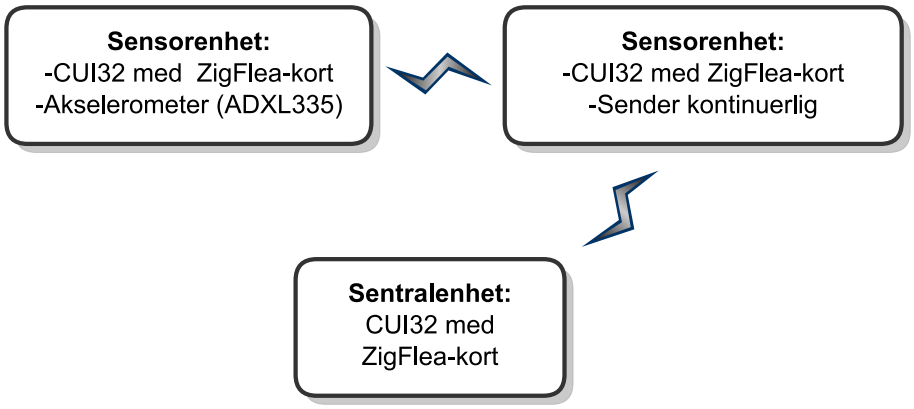
3.4.5 Oppsett 5 - Stjernenettverk med hyppig avlesing

I dette oppsettet skal jeg teste hvordan CUI32 og StickOS fungerer i et stjernenettverk. Ulike nettverkstopologier er beskrevet nærmere i avsnitt 2.5.2. Kort fortalt er et stjernenettverk et nettverk hvor flere sensorenheter sender data til den samme hovedenheten.

Siden det viktigste med dette oppsettet er å teste nettverkstopologien og ikke sensoroppsettet, har jeg brukt sensoroppsettet fra oppsett 2 (avsnitt 3.4.2) til den ene sensorenheten. Denne enheten har altså et akselerometer, mens den andre kun sender en variabel som endres fra «1» til «0». Jeg har altså ingen sensorer på denne enheten. Dette gjør jeg for å ha minst mulig forstyrrende elementer, og setter dermed testing av selve nettverkstopologien i fokus. En slik måte og sette opp et Aktiv Musikk-system på kan være nyttig hvis sensorene skal plasseres på forskjellige steder på kroppen. Man kan for eksempel ha en RFID-leser i skoen for å observere hvor i rommet brukeren befinner seg. Samtidig kan et akselerometer registrere bevegelse av en arm eller annen kroppsdel. Figur 3.21 viser hvordan dette er koblet sammen til et stjernenettverk.

3.4.6 Oppsett 6 - Punkt-til-punkt nettverk

Dette oppsettet skal på samme måte som oppsett 5, teste en bestemt nettverkstopologi. Ulike nettverkstopologier er nærmere beskrevet i avsnitt 2.5.2. Her er det punkt-til-punkt nettverk som blir testet, og figur 3.22 viser hvordan dette er koblet sammen. Her er sensorenhetene koblet sammen for å «samle» alle sensordata, før de blir sendt samlet til sentralenheten. For å kunne sam-



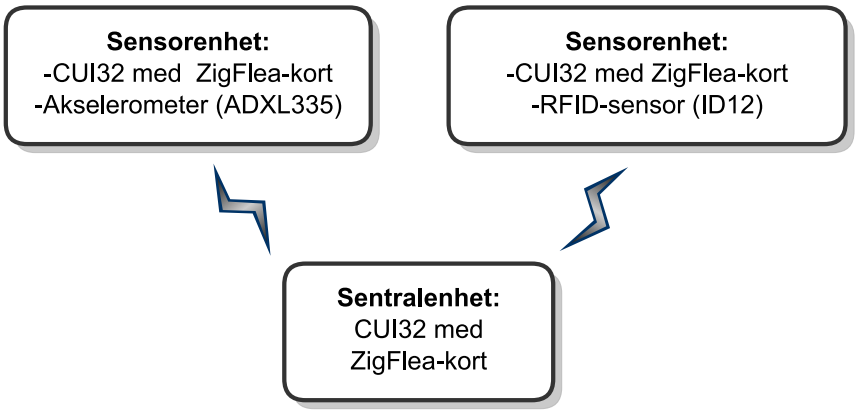
Figur 3.22: Tilkobling av sensorer og sammenkobling av sensorenheter og sentralenhet for oppsett 6.

menligne best mulig, bruker jeg samme oppsett på sensorenhet som for oppsett 5. Det vil si akselerometer på den ene sensorenheten, og kun en endrende variabel på den andre enheten. Dette for å kunne sammenligne sjernenettverket med punkt-til-punkt nettverket på en best mulig måte.

En slik sammenkobling av oppsettet sørger for at sensorenhetene får mulighet til å kommunisere. Dette kan være nyttig for å avlaste sentralenheten. Det er ikke nødvendigvis all informasjon som må sendes til denne. Allikevel er et slik oppsett avhengig av at sensorenheten i midten skifter raskt mellom sending og mottak. Det blir altså et av de viktigste punktene å teste for dette oppsettet.

3.4.7 Oppsett 7 - Stjernenettverk med sjelden avlesing

I dette oppsettet ønsker jeg på samme måte som for oppsett 5 og teste et stjernenettverk. For oppsett 5 er det slik at begge sensorenhetene sender data kontinuerlig. I dette oppsettet ønsker jeg å teste hvordan det fungerer hvis den ene sender kontinuerlig, mens den andre kun sender sporadisk. Jeg har derfor brukt en RFID-leser på den ene sensorenheten. På den andre sensorenheten bruker jeg akselerometeret. Denne vil sende data kontinuerlig, på samme måte som fra oppsett 2 og 5. Jeg får altså testet hvordan avbrudd fra en sensorenhet påvirker sending av kontinuerlige data fra en annen sensorenhet. Figur 3.23 viser hvordan dette oppsettet er koblet opp.



Figur 3.23: Tilkobling av sensorer og sammenkobling av sensorenheter og sentralenhet for oppsett 7.

Kapittel 4

Resultater

I dette kapitlet vil jeg beskrive hvordan de ulike sensoroppsettene fungerer og hva som er fordeler og ulemper med de. Jeg vil generelt vektlegge hastighet, stabilitet, ulike nettverkstopologier og hvor godt de egner seg til et Aktiv Musikksystem. Spesielt vil jeg vektlegge egenskapene som er beskrevet i tabell 3.2. Videre i denne teksten vil jeg bruke *sensordata* om data som kommer fra en sensor. En sensor kan gi ut flere sensordata. Orienteringssensoren kan for eksempel sende opp til 15 ulike sensordata, og hver av disse blir puttet i hver sin fjernstyrte variabel i StickOS. Disse variablene er bestemt til å være 32 biter. En *runde* er en gjennomkjøring av kjøreløkken, typisk while-løkke. Altså kodebiten som finnes mellom `while 1 do` og `endwhile` i kodeeksemplene.

4.1 Oppsett 1 - Orienteringssensor, avstandssensor og RFID

Etter å ha implementert oppsett 1 ser vi av tabell 4.1 og figur 4.1 at det tar ca $652,5ms$ å lese inn og overføre alle 16 sensordataene. Som nevnt i avsnitt 3.4.1 er dette 15 fra orienteringsseensoren og en fra avstandssensoren. Dette gir oss en oppdateringsfrekvens $f = \frac{1}{652,5ms} = 1,53Hz$. Om vi begrenser oss til å sende over kun 3 sensordata ser vi videre at vi får en oppdateringsfrekvens $f = \frac{1}{120,4ms} = 8,31Hz$. Dette betyr at vi har en del og hente på å spesifisere hvilke sensordata vi trenger for øyeblikket. Videre ser vi at sammenhengen mellom antall sensordata som sendes og tid per runde er omtrent helt lineær.

Når ingen av sensordataene blir overført, ser vi at programmet bruker $8,9ms$. Dette er avlesing av UART og lagring av disse verdiene lokalt. I en praktisk situasjon ville man sannsynligvis ikke bedt om data fra sensoren når ikke noe skal sendes, men i denne sammenheng er det mer interessant og se hvor lang tid dette faktisk tar. I tillegg er det mulig å sende beskjed til sensoren, slik at kun

dataene man trenger blir sendt fra sensor til CUI32. Dette vil man sannsynligvis spare litt tid på, men om det blir gjort ofte vil det også ta litt tid og gi beskjed til sensoren hvilke data man vil ha.

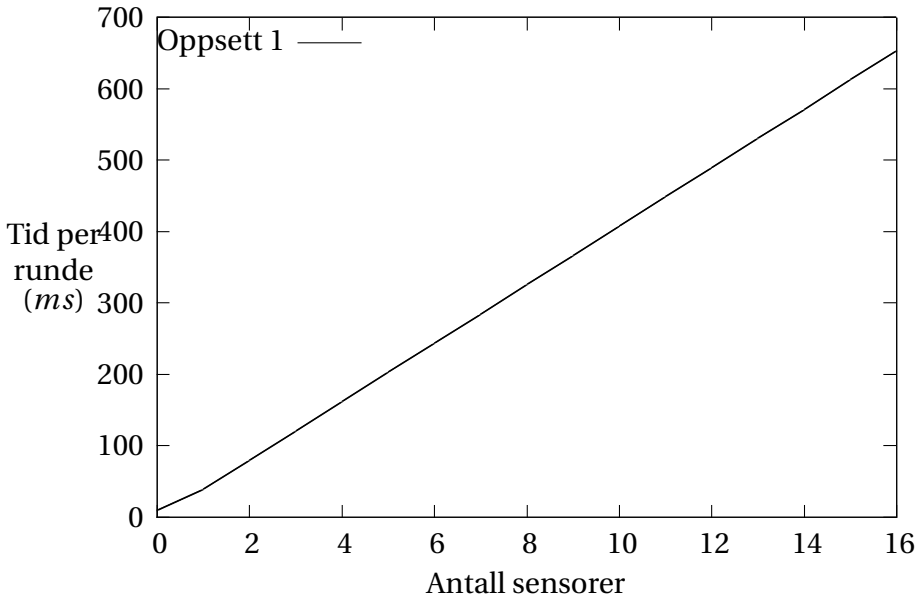
Antall sensordata	Tid per runde	Oppdateringsfrekvens	Tid per sensor
16	$652,5ms$	$1,53Hz$	$40,8ms$
15	$611,8ms$	$1,63Hz$	$40,8ms$
14	$569,9ms$	$1,75Hz$	$40,7ms$
13	$529,6ms$	$1,89Hz$	$40,7ms$
12	$488,7ms$	$2,04Hz$	$40,7ms$
11	$448,0ms$	$2,23Hz$	$40,7ms$
10	$406,7ms$	$2,46Hz$	$40,7ms$
9	$365,8ms$	$2,73Hz$	$40,6ms$
8	$325,1ms$	$3,08Hz$	$40,6ms$
7	$283,6ms$	$3,53Hz$	$40,5ms$
6	$243,2ms$	$4,11Hz$	$40,5ms$
5	$202,2ms$	$4,95Hz$	$40,4ms$
4	$161,3ms$	$6,20Hz$	$40,3ms$
3	$120,4ms$	$8,31Hz$	$40,1ms$
2	$79,1ms$	$12,6Hz$	$39,6ms$
1	$38,7ms$	$25,84Hz$	$38,7ms$
0	$8,9ms$	$112,36Hz$	-

Tabell 4.1: Sammenheng mellom antall sensordata som sendes og hastighet for oppsett 1.

Når det gjelder stabilitet har systemet en tendens til å henge seg opp når RFID-leseren gjør et avbrudd. Dette skjer bare så lenge data blir mottatt på den andre UART-porten. Problemet ser ut til å løse seg hvis man legger inn en pause på noen millisekunder etter hver runde. Dette gjør at vi får et system som er hakket tregere, men mer stabilt.

4.2 Oppsett 2 - Akselerometer

Som vi ser av tabell 4.2 og figur 4.2 er forskjellen i tid per sensordata minimal i forhold til oppsett 1. Vi får her en oppdateringsfrekvens $f = \frac{1}{120,1ms} = 8,33Hz$ når vi mottar alle tre sensordataene. Mot $8,31Hz$ for tre sensorer med oppsett 1. Dette sier oss at det er den trådløse overføring som tar mest tid, ikke avlesing av sensoren lokalt på sensorenheten. I tillegg bruker programmet bare $0,8ms$ på en runde når ingen av sensordataene blir sendt. Dette er fordi de analoge



Figur 4.1: Resultater oppsett 1. Tid per runde som funksjon av antall sensordata

inngangene heller ikke blir lest av, og vi slipper å bruke ADC-en. Om inngangene kun blir lest og lagt i en lokal variabel, uten å bli overført trådløst, bruker programmet ca $2ms$ pr runde.

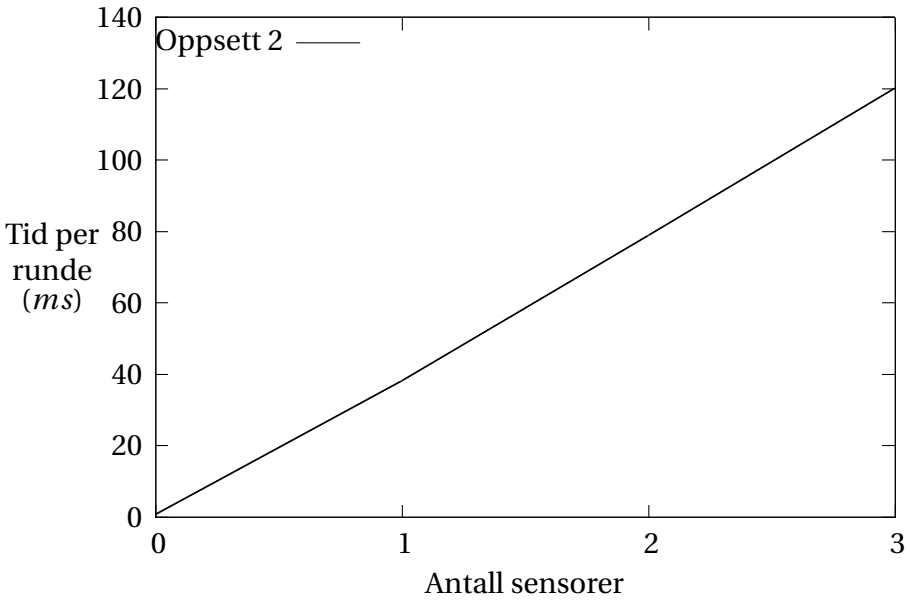
Antall sensordata	Tid/runde	Oppdateringsfrekvens	Tid/sensordata
3	$120,1ms$	$8,33Hz$	$40,0ms$
2	$78,9ms$	$12,67Hz$	$39,5ms$
1	$38,1ms$	$26,25Hz$	$38,1ms$
0	$0,8ms$	$1250Hz$	-

Tabell 4.2: Sammenheng mellom antall variabler som sendes og hastighet for oppsett 2.

Stabiliteten til dette oppsettet er utmerket. Her trenger vi ikke legge inn pauser som i oppsett 1. I tillegg kobles enhetene raskt sammen igjen hvis en av de har vært utenfor rekkevidden til transceiveren.

4.3 Oppsett 3 - Større forflytninger

I dette oppsettet er ikke hastighet spesielt kritisk. Om en RFID tag kommer fram noen millisekunder etter avlesing på sensorenheten vil ikke det merkes. På samme måte endrer ikke pulsen seg raskt nok til å forsvare en oppdatering



Figur 4.2: Resultater oppsett 2. Tid per runde som funksjon av antall sensordata.

flere ganger i sekundet. En overføring av data annethvert sekund viser seg å være mer enn tilstrekkelig til å holde oversikt over oppdatert puls. Stabiliteten er veldig god. Alle RFID målinger blir lest og overført riktig, og jeg opplevde ikke at programmet sluttet å kjøre. Dette systemet egner seg godt til å bruke i et Aktiv Musikk-system, med tanke på at hastigheten og stabiliteten som nevnt er god nok. I tillegg vil man både kunne registrere at brukeren flytter seg fra sted til sted ved hjelp av RFID, i tillegg til at man kan måle hvor mye aktivitet brukeren er i på andre måter ved hjelp av pulsmåleren.

4.4 Oppsett 4 - RFID

For dette oppsettet er det to tider vi er ute etter. Det er tiden sensorenheten bruker på å lese data fra RFID-sensoren, og hvor lang tid sensorenheten bruker på å overføre denne tagen til sentralenheten. Dette testes ved å skrive ut total kjøretid i ms når:

- lesing fra sensor starter,
- når lesing er ferdig,
- og når overføring til sentralenhet er ferdig.

Deretter målte jeg disse tidene ti ganger og regnet ut gjennomsnittet. Tiden viser seg å være $16ms$ for lesing av tagen, og $408ms$ for trådløs overføring av hele RFID-tagen. Siden RFID-tagen består av ti tegn, blir dette sendt som ti variabler i StickOS. Det vil si at tiden det tar å sende ett tegn/variabel blir omtrent den samme som vi har målt for tidligere oppsett. Altså $\frac{408ms}{10variabler} = \frac{40,8ms}{variabel}$.

4.5 Oppsett 5 - Stjernenettverk med hyppig avlesing

Når begge sensorenhetene forsøker å sende data kontinuerlig i stjernenettverket, stopper en av enhetene opp. Den kommer rett og slett ikke igjennom, siden sentralenheten er opptatt med å motta data fra den andre sensorenheten. Det er altså bare den ene sensorenheten som får sendt sine data. Det viser seg at dette kan løses hvis dataene ikke sendes kontinuerlig. Det vil si at man setter inn en pause i «while»-løkken. Dette kan gjøres ved hjelp av kommandoen `sleep` *tid* `(s|ms|us)`. Jeg forsøker å øke denne pausen gradvis. Det holder ikke å øke den til $50ms$. Derimot, hvis den økes til $100ms$ får begge sensorenheter sendt data. Det går allikevel i «rykk og napp», og det er ikke gitt at sensorenhetene får sende annenhver gang. Som regel blir det altså slik at den ene enheten får sendt oftere enn den andre. Dermed er det heller ikke lett å måle gjennomsnittstiden det tar å overføre en variabel. Jeg tar derfor et gjennomsnitt av 20 runder med sensordata (tilsvarer $20 \cdot 3 = 60$ sensordata). Dette gjøres flere ganger, slik at jeg kan bruke gjennomsnittet av dette igjen. Resultatene fra testing med pauser på $100ms$, $150ms$ og $200ms$ kan sees i tabell 4.3. Her sammenlignes også tiden som brukes i stjernettverket, opp mot tiden som brukes hvis kun den ene sensorenheten sender data. Vi ser at *forskjellen* til tiden avtar, ettersom pausen øker. Dette betyr at stjernettverket lager mindre konflikt ettersom pausen øker, noe som er naturlig. Hva som er den optimale pausen kommer an på hva man skal bruke systemet til. Om det er viktigst at data kommer fram fortest mulig vil $100ms$ være det beste. Derimot, hvis det er viktigere at data kommer annenhver gang bør man øke pausen til $200ms$ eller mer. Man får da et tregere, men mer stabilt system.

4.6 Oppsett 6 - Punkt-til-punkt nettverk

Dette oppsettet får noe av det samme problemet som forrige oppsett. Sensorenheten i midten får problemer med å både sende og motta data samtidig. Her er problemet enda større enn i forrige oppsett. Det virker til å være helt tilfeldig om denne enheten mottar eller sender data. Det kan derfor gå lang tid mellom hver gang et fullstendig datasett blir sendt til sentralenheten. Det kan se ut for

Pause	Stjernenettverk		En sensorenhet		Forskjell
	Alle sensor-data	En sensor-data	Alle sensor-data	En sensor-data	
0	-	-	$121,4ms$	$40,4ms$	-
$100ms$	$358,6ms$	$119,5ms$	$192,0ms$	$64,0ms$	$55,5ms$
$150ms$	$375,7ms$	$125,2ms$	$242,0ms$	$80,7ms$	$44,5ms$
$200ms$	$417,9ms$	$139,3ms$	$295,0ms$	$98,3ms$	$41,0ms$

Tabell 4.3: Sammenheng mellom lengde på pause per runde, og hvor lang tid det tar å overføre en variabel. Sammenligning av stjernenettverk med to sensorenheter, og nettverk med kun en sensorenhet for oppsett 5.

at problemet avtar noe om man legger inn pauser per runde, slik jeg gjorde for oppsett 5. Allikevel varierer målingene så mye, at jeg har valgt å kun ta med en av de beste gjennomsnittsmålingene, når jeg la inn en pause på $300ms$. Systemet bruker da $455ms$ per variabel som overføres. Som vi ser er dette en mye lengre tid, enn det som er tilfellet for stjernenettverket.

4.7 Oppsett 7 - Stjernenettverk med sjelden avlesing

Det viser seg at RFID-leseren bruker nøyaktig like lang tid på lesing og sending som for oppsett 4. Det vil si $16ms$ på lesing av sensor og $408ms$ på trådløs overføring. På samme måte som for det andre stjernenettverkoppsettet (avsnitt 4.5), stopper overføringen som allerede er i gang, når tagen blir sendt. Dermed vil overføringen av akselerometerdataene bli forsinket med i overkant av $408ms$ når en ny RFID-tag blir lest.

Kapittel 5

Diskusjon

Dette avsnittet tar for seg vurdering og diskusjon av resultatene fra avsnitt 4. Hovedpunktene for denne diskusjonen er hastighet, brukervennlighet, nettverkstopologi og stabilitet. Til slutt i kapittelet kommer en sammenligning med systemene fra avsnitt 2.9

5.1 Hastighet

Resultatene fra oppsett 1 og 2 viser oss at vi har en hastighet på like over $25Hz$ for hver enkelt sensordata som skal sendes. Vi ser også at sammenhengen mellom antall sensordata, og tiden det tar å sende over alle sensordataene, er tilnærmet lineær. Det betyr at dersom vi øker antall sensordata som skal sendes, øker vi tiden det tar å sende over alle dataene med omtrent konstant tid per sensordata. Overføringshastigheten per enkelt sensordata er mer eller mindre uendret $(38,7ms - 40,8ms)$, uavhengig av antall data som sendes. Hvis sensordataene skal brukes som rådata til bevegelsesanalyse, bør man ha en oppdateringsfrekvens på $50 - 60Hz$. Fra tabell 4.2 ser vi at 26,25Hz er det høyeste resultatet, og det er for kun en sensordata. Om man skal brukes akselerometeret til bevegelsesanalyse vil man som regel trenge både x-, y- og z-akselerasjonen, og dermed få en oppdateringsfrekvens på omkring $8,33Hz$ (tabell 4.2). Med andre ord er ikke overføringshastigheten god nok til slik bevegelsesanalyse.

På den andre siden er det på langt nær alle Aktiv Musikk-systemer som bruker bevegelsesanalyse som er avhengig av hastigheter på mer enn $25Hz$. Dette er oppsett 3 et godt eksempel på. Her vil man kun trenge å overføre data når en ny RFID tag blir lest, og annethvert sekund med data om oppdatert puls. Her blir data fra pulsmåleren lest av kontinuerlig lokalt på sensorenheten. Overføring til sentralenheten trenger ikke skje så ofte, siden pulsen ikke endrer seg så raskt, som for eksempel data fra akselerometeret.

Beregning av puls på sensorenheten et eksempel på at analyse lokalt på sensorenheten kan gjøre det trådløse overføringsbehovet mindre. På samme måte kan det gjøres enkle beregninger på bevegelsesdata, og deretter sende disse beregningene så ofte som mulig. Man kan for eksempel være interessert i gjennomsnittlig bevegelsesmengden de siste x sekundene, enten totalt eller i de forskjellige retningene. Denne beregningen kan sensorenheten gjøre, og deretter sende til sentralenhet annethvert sekund. På denne måten blir sensordataene registrert kontinuerlig, men overføringsbehovet blir mye mindre.

I StickOS kan ikke de fjernstyrte variablene deklareres som noe annet enn 32-bits variabler. Hvis en sensordata er mindre enn 32-bit, vil det si at unødvendig mye data blir overført. Dette kan løses ved at man sier at hver enkelt sensordata kun kan være for eksempel 16 bits. På denne måten kan man lagre 2 sensordata i hver 32-bits variabel. Dette gjøres ved at de 16 mest signifikante bitene i variabelen lagrer en sensordata, mens de 16 minst signifikante lagrer en annen. Datamengden som må sendes blir altså halvert. Dette vil føre med seg en del ekstra programmering og bitfikling. Sensorenheten må pakke sensordataene inn i én variabel, og sentralenheten må pakke de ut igjen. Dette vil gi litt mer prosessering av dataene før de sendes, men vil gi en samlet tidsgevinst. Man kan selvfølgelig også lagre hver sensordata som 8 bit eller enda mindre. Hva som er mest hensiktsmessig å bruke, vil avhenge av oppløsningen på sensordataene. De analoge inngangene på CUI32 varierer mellom $0V$ og $3300mV$. Siden det trengs 12 bits for å lagre tallet 3300, vil nok 16 bits pr sensordata være en god løsning her. RFID tagene består av 10 tegn på 32 bit hver, så her er det ikke så mye og hente, mens orienteringssensoren sender 16-bits sensordata. Datamengden som sendes vil altså kunne halveres både for sensorene som sender analogt og orienteringssensoren.

StickOS bør generelt endres til å kunne håndtere variabler på andre størrelser enn 32 biter. Da kunne man enkelt deklarert 16-biters variabler, og man hadde sluppet å overføre unødvendige datamengder. Dette vil vært en fordel for å opprettholde brukervennligheten til StickOS. Brukere med liten programmeringserfaring vil muligens ikke være komfortable med å programmere på bitnivå. I tillegg til dette er ZigFlea-protokollen i StickOS slik, at hver enkelt variabel sendes som en pakke. Dette betyr at sendingen startes og stoppes for hver enkelt variabel. Her bør det gjøres forbedringer i StickOS, slik at en større mengde biter kan sendes i hver pakke. Dette vil sannsynligvis øke overføringshastigheten betydelig.

Ved å ta hensyn til og endre det som er beskrevet i avsnittene over, vil man kunne øke hastigheten en god del. Allikevel spørs det om StickOS og ZigFlea-protokollen vil kunne klare hastigheter som er gode nok for overføring av rådata til hastighetskrevende bevegelsesanalyse, uten å måtte gjøre enda betydeligere endringer.

5.2 Brukervennlighet

En av hovedårsakene til at jeg valgte CUI32 og StickOS som plattform var at jeg ville teste brukervennligheten. Som nevnt tidligere er det raskt å komme i gang med. Som man kan se av kodeeksemplene er det også få linjer med kode som skal til før sending av data er i gang. Dette gjør at man ikke trenger å være en erfaren programmerer for å få til dette. Allikevel er det greit å ha en viss erfaring med for eksempel løkker og if-setninger.

De analoge inngangene på CUI32 er lett tilgjengelig, med forsyningsspenning og jord like ved siden av selve sensorinngangen. Disse er også dimensjonert slik at man kan bruke phidgets kontakter direkte. Dette gjør tilkoblingen av analoge sensorer rask og enkel. For å finne de digitale inngangene kan man skrive `help pins` i terminalemulatoren for å få opp en liste over pinnene. På denne måten er det enkelt og også finne hvilke pinner som hører til UART, SPI osv.

På den andre siden er det et problem at pauser må legges inn i kjøreløkken. En bruker som ikke har erfaring med programmering, vil sannsynligvis ha større problemer med å forstå hvor disse skal være. Dette vil føre til frustrasjon, og en lite brukervennlig opplevelse.

5.3 Nettverkstopologi

Når nettverket kobles opp med flere sensorenheter får StickOS enda større hastighetsproblemer. Det er forventet, men ikke at problemene skulle bli såpass store. Det kan virke som at synkronisering mellom to enheter som forsøker å sende samtidig, ikke er implementert i StickOS. Dette problemet er naturlig nok størst når data skal sendes og mottas kontinuerlig. Dette er tilfellet for oppsett 5 og 6 (avsnitt 4.5 og 4.6). Her må det settes inn pauser i kjøringen for å «slippe til» den andre sensorenheten. Det viser seg også at stjernenettverket fungerer mye bedre enn punkt-til-punkt nettverket.

StickOS har få muligheter til å konfigurere den trådløse overføringen, og det kan bli vanskelig å finne en rask løsning på dette. Man kan tenke seg en løsning hvor sensorenheter sjekker om den andre sender, og deretter sender annenhver gang. Dette vil på den andre side føre til datatrafikk mellom sensorenhetene, noe som ikke viste seg å være spesielt heldig. En måte å løse problemet på er å omprogrammere den trådløse delen av StickOS. Med dette mener jeg at det kan legges inn bedre synkronisering fra bunn av. I tillegg bør det være mulig å bestemme hvilke enhet man vil motta data fra. På denne måten vil det være mulig å styre det slik at man først mottar data fra sensorenhet 1, deretter fra sensorenhet 2 for eksempel.

På den andre siden er det fullt mulig å bruke stjernenettverk til systemer som ikke trenger å reagere så raskt. Dette er oppsett 7 fra avsnitt 4.7 et godt eksempel på. Her får akselerometerdataene små forsinkelser når nye RFID-tager blir lest. Bortsett fra det blir data sendt så kjapt som det er mulig med StickOS.

5.4 Stabilitet

Oppsettene er stort sett stabile når man får lagt inn riktig størrelse på pauser per runde. Den eneste formen for ustabilitet som er registrert skjer som nevnt i oppsett 1, med lesing fra to UART-porter. Her er det muligens en konflikt mellom disse som er problemet.

Hvis enhetene kommer utenfor rekkevidden til transceiverne, kobles de raskt sammen når de er innenfor igjen. Dette gjør at oppsettene virker veldig stabile og robuste. Her har ZigFlea og ZigBee en fordel framfor Bluetooth.

Når man forsøker å koble sammen flere enn to enheter, blir stabiliteten dramatisk dårligere. Det er ingen ordentlig synkronisering mellom enhetene, og det virker tilfeldig hvilke av enhetene som får sende data først og sist. Selv om systemet sjeldent henger seg helt, forventer man at data skal sendes og mottas med bedre synkronisering enn dette.

5.5 Sammenligning med andre systemer

Jeg vil i dette avsnittet sammenligne systemene jeg har beskrevet i avsnitt 2.9, med CUI32 og StickOS. Jeg vil i hovedsak se på hastighet, brukervennlighet, pris og hvordan systemene fungerer med mer enn en sensorenhet. Stabilitet er vanskeligere å sammenligne, og er ikke like godt beskrevet i de ulike artiklene. Jeg har derfor ikke et eget avsnitt for dette, men nevner det der det er naturlig. For systemet til Renton [23] vil jeg bruke oppsettet med stjernenettverk som sammenligningsgrunnlag. Dette fordi det ligner mest på de andre.

5.5.1 Hastighet

Sammenlignet med de andre Aktiv Musikk-systemene fra avsnitt 2.9 er hastigheten lavest for CUI32 med StickOS. Sense/Stage oppgir $28,61ms$ som minste gjennomsnittstid for å overføre en pakke som varierer i størrelse fra 5 til 24 bytes. [3] De har da lagt brukt et tidsintervall på $25ms$, slik at selve overføringen kun tar $3,61ms$. Renton [23] skriver at hans system bruker $2,729ms$ på 3 bytes. CUI32 med StickOS bruker $38,1ms$ på å overføre en variabel på

$32 biter = 4 bytes$. Dette er betydelig saktere enn de andre systemene. KiiWii [1] prosjektet sier ikke noe om sine hastigheter.

5.5.2 Brukervennlighet

Når det gjelder brukervennlighet, bruker både Sense/Stage [3] og KiiWii [1] arduino, som er forholdsvis greit å sette seg inn i. Allikevel krever det installasjon av programvare og større kjennskap til programmering enn for StickOS. Renton [23] bruker Atmel mikrokontrollere. Disse programmeres i C som krever enda mer programmeringserfaring. CUI32 har StickOS installert og programmeres i forenklet Basic. Her kreves ingen annen programvare enn terminalemulator for å komme i gang. Læringskurven er derfor vesentlig slakere enn for de tre andre systemene.

5.5.3 Flere sensorenheter

Sense/Stage [3] har som mål at mange enheter skal kunne kobles sammen. De har testet med 10 enheter, og bruker gjennomsnittlig $72,385 ms$ på overføring. Da er tidsintervallene på $50 ms$, noe som vil si en reel overføfring på $22,385 ms$. Renton [23] bruker $10,03 ms$ når fire sensorenheter er koblet til. Han har ikke testet med flere enheter, siden han har satt $10 ms$ som en øvre grense for overføringstid. CUI32 og StickOS har en overføringstid på hele $119,5 ms$ per fire byte, når to sensorenheter kobles til en sentralenhet. Den er derfor tregest av de tre. KiiWii [1] har ikke oppgitt om de har forsøkt å koble til mer enn en sensorenhet, men det er utenfor målet til deres prosjekt.

Kapittel 6

Konklusjon

Denne oppgaven har testet hvordan CUI32 og StickOS fungerer som plattform i et Aktiv Musikk-system. Det er lagt vekt på å teste brukervennlighet, hastighet, stabilitet og ulike nettverkstopologier. Dette er gjort ved å teste syv ulike oppsett av sensorenheten(e).

Som vi ser av resultatene i avsnitt 4 og diskusjonen i avsnitt 5, ser vi at CUI32 og StickOS ikke har en trådløs overføringshastighet som er god nok til Aktiv Musikk-systemer hvor hastighet er avgjørende. Hvis man skal bruke det til dette er man nødt til å gjøre endringer i kildekoden til StickOS. Man kan få sendt noe mer ved å bruke variabler på en tungvint måte, det vil si å legge flere sensordata inn i en variabel hvis det er mulig. Fra resultatene får man uansett ikke sendt mer enn omtrent 25 variabler i sekundet. Dette tilsvarer en oppdateringsfrekvens på omtrent $25Hz$, eller en hastighet på $25Hz \cdot 32bit = 800bit/s = 0,8kbit/s$. Systemer som krever en kontinuerlig strøm av for eksempel akselerometerdata vil trenge en høyere hastighet enn dette. Det vil si at CUI32 og StickOS egner seg best for systemer hvor stabilitet og brukervennlighet er det viktigste, mens hastighet er nedprioritert. Dette kan være systemer hvor musikken endres langsommere, eller at endringer skjer på bakgrunn av hvor man befinner seg i et rom (RFID). Slike systemer vil være mye raskere og mer brukervennlig å sette opp med StickOS, enn om man skulle brukt for eksempel programmeringsspråket C eller Arduino.

Når det gjelder nettverkstopologi er oppsettet best egnet til å kun ha en sensorenhet og en sentralenhet. Dette gjelder i hvert fall om man er opptatt av hastighet. Om man allikevel skal bruke flere sensorenheter, viser det seg at stjernenettverk er bedre egnet enn punkt-til-punkt nettverk. I tillegg bør en av sensorenhetene i stjernenettverket kun sende data sporadisk. Hvis begge (eller flere) sensorenhetene skal sende samtidig må det legges inn pauser i kjøreløkkene, slik at ikke en eller flere enheter stopper å kjøre.

Konklusjonen er at CUI32 og StickOS fungerer utmerket til systemer hvor

kravet til hastighet på trådløs overføring er lavere enn $0,8 kbit/s$. Hastighet på lesing av sensorer er god. I tillegg er dette en solid plattform med høy brukervennlighet. Hvis man i tillegg har litt programmeringserfaring vil man raskt kunne utvikle mange ulike prototyper. Det viser seg også at strømforbruket er lavt og at systemet stort sett jobber stabilt.

Kapittel 7

Videre arbeid

Det største problemet med plattformen slik den er nå, er hastigheten på den trådløse overføringen. Dette vil derfor være et naturlig utgangspunkt for videre utvikling. Her bør man videreutvikle kildekoden til StickOS. Man kan se på den delen av koden som brukes til trådløs overføring, og forbedre denne. Noen forslag er nevnt i avsnitt 5.1. Dette går på og kunne deklarere fjernstyrte variabler som er tilpasset størrelsen på sensordataene som skal sendes. I tillegg bør større mengder data kunne sendes i en pakke. Ikke slik som nå, at sending startes og stoppes for hver variabel. Dette vil kunne øke hastigheten betydelig. I tillegg til dette vil det være en fordel å implementere støtte for flyttall i StickOS, slik at flere beregninger kan gjøres direkte.

Det er selvfølgelig en mulighet og droppe StickOS, og heller skrive kode i C som brukes på CUI32. Da vil man få høyere hastigheter, siden man da kan konsentrere seg kun om ett oppsett og spesialtilpasse kode til dette. Man trenger med andre ord ikke skrive et helt operativsystem i tillegg, og ta hensyn til at dette trenger en del ressurser for å kjøre. På den andre siden vil dette gi en mye dårligere brukervennlighet. Ett alternativ kan være gumstix eller arduino som er beskrevet i avsnitt 2.8.1 og 2.8.2. Det vil alltid være en vurdering basert på hva man trenger til hvert enkelt system som skal lages.

I tillegg til mulige forbedringer som er nevnt tidligere i avsnitet, bør systemene testes av reelle brukere. Man vil da få en mer fullstendig oppfatning av hvordan systemet fungerer i sitt virkelige miljø. I tillegg kan man bruke intervjuer og spørreundersøkelser for å få tilbakemeldinger som kan forbedre systemet ytterligere. Disse tilbakemeldinger kan komme både fra de som utvikler egne prototyper, og brukere av de ferdige systemene.

Bibliografi

[1] Nabil Aounallah, Austin Grossman, David Liu, Milton Villeda og Bob Yu. Wireless remote based music synthesizer. 2009. http://kiiwiiproject.googlecode.com/files/EE464%20-%20KiiWii%20Final%20Report.pdf.

[2] IEEE Standard Association. Ieee 802.15.4d. http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf, 2009.

[3] Marije A.J. Baalman, Vincent de Belleval, Christopher L. Salter, Joseph Malloch, Joseph Thibodeau og Marcelo M. Wanderley. Sense/stage - low cost, open source wireless sensor infrastructure for live performance and interactive, real-time environments. *ICMC*, juni 2010. http://www.cse.ucsd.edu/classes/fa08/cse237a/topicresearch/jkooker_tr_report.pdf.

[4] Anne Bøgh og Mimmi Larsson. Sensor devices - physiological grab- and wearable wired and wireless sensors. http://re-new.org/research/sum/sensors/, oktober 2009.

[5] CPUstick. Cpustick and stickos – embedded systems made easy. http://cpustick.com/.

[6] Analog Devices. Adxl335. http://www.analog.com/en/sensors/inertial-sensors/adxl335/products/product.html.

[7] Richard O. Duda, Peter E. Hart og David G. Stork. *Pattern Classification*. John Wiley & Sons, Inc, 2 utgave, 2001.

[8] Jacob Fraden. *Handbook of Modern Sensors*. Springer, 3 utgave, 2004.

[9] Ørjan G. Martinsen. *PC-basert instrumentering og mikrokontrollere*. Gyldendal, 2006.

[10] V. Gallese, L. Fadiga, L. Fogassi og G. Rizzolatti. Action recognition in the premotor cortex. *Brain*, 119(2):593–609, 1996.

[11] James J. Gibsons. *The Ecological Approach to Visual Perception.* New York: Houghton-Mifflin, 1979.

[12] B. L. Giordano. *Sound source perception in impact sounds.* Doktorgradsoppgave, University of Padova, 2005.

[13] Gumstix. Gumstix. `www.gumstix.com`.

[14] Bruno Herbelin. Bio-signal device prototype 2. september 2010.

[15] Mats Høvin, Lena Mariann Garder, Rolf Inge Godøy, Jim Tørresen og Alexander R. Jensenius. Sensing music-related actions. Prosjektbeskrivelse.

[16] ID Innovations. Id series datasheet. `http://www.sparkfun.com/datasheets/Sensors/ID-12-Datasheet.pdf`, mars 2005.

[17] Alexander Refsum Jensenius. *ACTION - SOUND Developing Methods and Tools to Study Music-Related Body Movement.* Doktorgradsoppgave, Universitet i Oslo, 2007.

[18] John Kooker. 'bluetooth, zigbee, and wibree: A comparison of wpan technologies. `http://www.cse.ucsd.edu/classes/fa08/cse237a/topicresearch/jkooker_tr_report.pdf`, november 2008.

[19] A. M. Liberman og I. G. Mattingly. The motor theory of speech perception revised. *Cognition*, 21:1–36, oktober 1985.

[20] Microchip. Pic32mx440f512h. `http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en535591`.

[21] Tom M. Mitchell. *Machine Learning.* The McGraw-Hill Companies, Inc, 1997.

[22] Dan Overholt. cui32. `http://code.google.com/p/cui32/`, 2010.

[23] Eirik Renton. Collecting sensor data for active music. Hovedfagsoppgave, Universitetet i Oslo, juni 2009.

[24] CH Robotics. Chr-6dm attitude and heading reference system. `http://www.chrobotics.com/docs/chr6dm_datasheet.pdf`.

[25] Freescale semiconductor. Compact integrated antenna. `http://www.freescale.com/files/rf_if/doc/app_note/AN2731.pdf`, juli 2006.

[26] Freescale semiconductor. Mc13201. `http://cache.freescale.com/files/rf_if/doc/data_sheet/MC13201.pdf`, april 2008.

[27] Wiki.net. Openembedded. http://wiki.openembedded.net/index.php/Main_Page.

[28] Wikipedia. Radio-frequency identification. http://en.wikipedia.org/wiki/Radio-frequency_identification.

[29] Wikipedia.org. Arduino. http://en.wikipedia.org/wiki/Arduino.

[30] Wikipedia.org. Field-programmable gate array. http://en.wikipedia.org/wiki/Field-programmable_gate_array.

[31] Wikipedia.org. Ieee 802.15.4-2006. http://en.wikipedia.org/wiki/IEEE_802.15.4-2006.

[32] Wikipedia.org. Osi model. http://en.wikipedia.org/wiki/OSI_model.

[33] Wikipedia.org. Personal area network. http://en.wikipedia.org/wiki/Personal_area_network.

[34] Wikipedia.org. Serial peripheral interface bus. http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus.

[35] www.AsciiTable.com. Ascii table and description. http://www.asciitable.com/.

Tillegg A

Beskrivelse av film med demonstrasjon

Til denne oppgaven er det laget en film som demonstrerer bruk av CUI32 og StickOS med noen ulike sensorer. Den følger med på cd under innlevering, og finnes også på `http://www.youtube.com/watch?v=vCMdFizucng`. Her kommer en beskrivelse av de ulike delene av filmen, minutt for minutt:

- **0:00 - 0:47:** Viser CUI32- og ZigFlea-kortet, og hvordan de kobles sammen.
- **0:48 - 1:59:** Viser hvordan `nodeid 1` programmeres trådløst, ved hjelp av `nodeid 2`. Deretter kjøres programmet fra `nodeid 1` ved hjelp av kommandoen `run`.
- **2:00 - 2:20:** Viser tilkobling av akselerometer.
- **2:21 - 4:30:** Viser programmering av enhetene slik at akselerometerdata sendes ved hjelp av fjernstyrte variabler. `Nodeid 1` (venstre) er sensorenhet, mens `nodeid 2` (høyre) er sentralenhet.
- **4:30 - 5:00:** Viser trådløst mottak av sensordata fra akselerometer.
- **5:00 - 6:30:** Viser demonstrasjon av RFID-leser og trådløs overføring av RFID-tag. Ved 5:45 aktiveres `autorun` på sensorenheten, og den kobles bort fra datamaskinen. Deretter leses flere RFID-tager. Ved 6:09 kobles batteri fra og deretter til igjen. Dette for å demonstrere hvor raskt sensorenheten kobler seg til sentralenheten på nytt.
- **6:30 - 8:06** Viser demonstrasjon av orienteringssensoren, RFID-leser og Max/MSP patch. Patchen starter kjøring av program på sentralenheten,

mottar informasjon fra sensorenheten om hvilke sensorer som er koblet til. Deretter kan man skru på og av mottak av yaw, pitch og roll grader, ved å trykke på henholdsvis «1» og «0». Deretter leser jeg av noen RFID-tager.

Tillegg B

Kode til sentralenhet.

```
dim testOn
dim i
dim n_sensors
let n_sensors = 16
dim r[n_sensors]
dim rfid[10] as byte
dim sensors_on
dim a as remote on nodeid 1
dim text_new$[16]
dim text_old$[16]
dim sensorInfo$[20]
dim needInfo as remote on nodeid 1

while 1 do
        input text_new$
        if text_new$ != text_old$ then
                gosub atob
        endif
        print "A",r
        print "B",raw rfid
        if sensorInfo$ == "" then
                let needInfo = 1
        elseif needInfo == 1 || needInfo == -1 then
                let needInfo = 0
        endif
        print "I",raw sensorInfo
        sleep 300ms
endwhile
```

```
sub atob
        dim bit
        let bit = 1
        let i = n_sensors-1
        rem -- Update locale sensors_on first
        while i >= 0 do
                if text_new[i] == '1' then
                        let sensors_on = (sensors_on | bit)
                else
                        let sensors_on = (sensors_on & ~bit)
                endif
                let bit = bit*2
                let i = i - 1
        endwhile
        print "sensors_on",hex sensors_on
        rem -- Update remote variable a
        while a != sensors_on || a == -1 do
                let a = sensors_on
                sleep 50ms
        endwhile
        vprint text_old$ = text_new$
endsub
```

Tillegg C

Kode til sensorenhet på oppsett 1

```
dim sensorNames$[20]
let sensorNames$ = "chr␣sonar␣rfid"

rem -- chr-6d variables:
dim u2sta at address 0xbf806010
dim byte as byte
dim lest
dim snp
dim ns as byte
dim i as byte
dim data[32] as byte
configure uart 1 for 115200 baud 8 data no parity
dim ut[1] as byte
let ut[0] = ut#
dim s[15]

rem -- Analog sensors
dim an0 as pin an0 for analog input

rem -- RFID variables:
dim b as byte
dim bytesread as byte
dim idread[10]
configure uart 2 for 9600 baud 8 data no parity

rem -- Remote variables
dim testOn as remote on nodeid 2
```

```
dim r[16] as remote on nodeid 2
dim rfid[10] as remote on nodeid 2
dim a
dim sensorInfo[20] as remote on nodeid 2
dim needInfo

rem --Send sensor info
configure timer 0 for 2s
on timer 0 do gosub sendInfo

rem --Set silent mode--
gosub tx 0x81,ut
gosub rx
on uart 2 input do gosub rxid
while 1 do
        rem --Send chr-6d sensor data
        gosub tx 0x01,ut
        gosub rx
        gosub sendSensorData
        sleep 500ms
endwhile
sub rx
        let snp = 0
        uart 1 read byte
        if byte == 0x73 then
                uart 1 read byte
                if byte == 0x6e then
                        uart 1 read byte
                        if byte == 0x70 then
                                let snp = 1
                        else
                                return
                        endif
                else
                        return
                endif
        else
                return
        endif
        if snp == 1 then
                rem --Read packet type
```

```
        uart 1 read byte

        rem -- SENSOR DATA --
        if byte == 0xb7 then
                uart 1 read ns
                for i = 0 to ns-1
                        uart 1 read data[i]
                next
                rem TODO: check checksum
                uart 1 read byte
                uart 1 read byte

        rem -- COMMAND COMPLETE --
        elseif byte == 0xb0 then
                rem - Read N:
                uart 1 read byte
                rem - Read packet type completed:
                uart 1 read byte
                print "Command ",hex byte," complete"
                rem - Read checksum:
                uart 1 read byte
                uart 1 read byte
                return
        else
                print "Another package type.."
                return
        endif
endif
rem dim s[(ns-2)/2]
dim j
let j = 0
for i = 2 to ns-2 step 2
        rem -- Negative numbers
        if data[i] > 127 then
                let s[j] = ((data[i]<<8) | data[i+1]) - 65536
        rem -- Positive numbers
        else
                let s[j] = ((data[i]<<8) | data[i+1])
        endif
        let j = j + 1
next
```

```
        print s
endsub
sub tx type, data
        dim i as byte
        dim packet[data[0]+6] as byte
        let packet[0] = 's'
        let packet[1] = 'n'
        let packet[2] = 'p'
        let packet[3] = type
        let packet[4] = data[0]-1
        for i = 0 to data[0] - 2
                let packet[i+4] = data[i+1]
        next
        dim checksum
        for i = 0 to data[0] + 3
                let checksum = checksum + packet[i]
        next
        let packet[data[0]+4] = (checksum>>8) & 0x0ff
        let packet[data[0]+5] = checksum & 0x0ff
        for i = 0 to data[0] + 5
                uart 1 write packet[i]
        next
endsub
sub rxid
        off uart 2 input
        uart 2 read b
        if b==0x2 then
                let bytesread = 0
                while bytesread <= 10 do
                        uart 2 read b
                        if bytesread < 10 then
                                if b == 0x2 || b == 0xd || b == 0xa || b ==
                                break
                                endif
                                let idread[bytesread] = b
                        elseif bytesread == 10 then
                                rem TODO: check checksum
                                uart 2 read b
                                uart 2 read b
                                uart 2 read b
                        endif
```

```
                        let bytesread = bytesread + 1
                endwhile
                print raw idread
        endif
        for i = 0 to idread#-1
                let rfid[i] = idread[i]
        next
        on uart 2 input do gosub rxid
endsub
sub sendSensorData
        print hex a
        if (a & 32768) != 0 then
                let r[0] = s[0]
        endif
        if (a & 16384) != 0 then
                let r[1] = s[1]
        endif
        if (a & 8192) != 0 then
                let r[2] = s[2]
        endif
        if (a & 4096) != 0 then
                let r[3] = s[3]
        endif
        if (a & 2048) != 0 then
                let r[4] = s[4]
        endif
        if (a & 1024) != 0 then
                let r[5] = s[5]
        endif
        if (a & 512) != 0 then
                let r[6] = s[6]
        endif
        if (a & 256) != 0 then
                let r[7] = s[7]
        endif
        if (a & 128) != 0 then
                let r[8] = s[8]
        endif
        if (a & 64) != 0 then
                let r[9] = s[9]
        endif
```

```
        if (a & 32) != 0 then
                let r[10] = s[10]
        endif
        if (a & 16) != 0 then
                let r[11] = s[11]
        endif
        if (a & 8) != 0 then
                let r[12] = s[12]
        endif
        if (a & 4) != 0 then
                let r[13] = s[13]
        endif
        if (a & 2) != 0 then
                let r[14] = s[14]
        endif
        if (a & 1) != 0 then
                let r[15] = an0
        endif
endsub
sub sendInfo
        let testOn = 0
        if testOn == -1 || needInfo == 0 then
                rem --Host off or don't need info
        else
                print "Sending␣sensorInfo"
                let sensorInfo$ = sensorNames$

        endif
endsub
```

Tillegg D

Kode til sensorenhet på oppsett 2

```
dim an0 as pin an0 for analog input
dim an1 as pin an1 for analog input
dim an2 as pin an2 for analog input

dim r[3] as remote on nodeid 2
dim test[3]

while 1 do
        print msecs
        let test[0] = an0
        let test[1] = an1
        let test[2] = an2
        print test
endwhile
```

Tillegg E

Kode til sensorenhet på oppsett 3

```
rem --- BVP variables ---
dim an1 as pin an1 for analog input
dim printet
dim lastmsecs
dim pulse
dim pulselimit
let pulselimit = 2500

rem --- RFID variables ---
dim i
dim b as byte
dim bytesread as byte
dim idread[10]
configure uart 2 for 9600 baud 8 data no parity
on uart 2 input do gosub rxid

rem --- Remote variables ---
dim rfid[10] as remote on nodeid 2
dim pulser as remote on nodeid 2

rem --- Timer interrupt for sending data
configure timer 0 for 2000ms
on timer 0 do gosub senddata

rem --- Whileloop ---
while 1 do
        if an1 > pulselimit && printet == 0 then
```

```
                let pulse = 60000/(msecs-lastmsecs)
                print "Puls",pulse
                let printet = 1
                let lastmsecs = msecs
        elseif an1 < pulselimit && printet == 1 then
                let printet = 0
        endif
        sleep 20 ms
endwhile

rem --- Sub for RFID ---
sub rxid
        off uart 2 input
        uart 2 read b
        if b==0x2 then
                let bytesread = 0
                while bytesread <= 10 do
                        uart 2 read b
                        if bytesread < 10 then
                                if b == 0x2 || b == 0xd || b == 0xa || b ==
                                break
                                endif
                                let idread[bytesread] = b
                        elseif bytesread == 10 then
                                uart 2 read b
                                uart 2 read b
                                uart 2 read b
                        endif
                        let bytesread = bytesread + 1
                endwhile
                print raw idread
                for i = 0 to idread#-1
                        let rfid[i] = idread[i]
                next
        endif
        on uart 2 input do gosub rxid
endsub

sub senddata
        rem -- Is pulse measured reasonable?
        if pulse > 50 && pulse < 220 then
```

```
                let pulser = pulse
        endif
endsub
```

Tillegg F

Kode til på oppsett 4

F.1 Sentralenhet

```
dim rfid[10]
while 1 do
    print raw rfid
    sleep 100 ms
endwhile
```

F.2 Sensorenhet

```
dim b as byte
dim bytesread as byte
dim idread[10]
dim i
dim rfid[10] as remote on nodeid 2
configure uart 2 for 9600 baud 8 data no parity
on uart 2 input do gosub rxid
while 1 do
endwhile
sub rxid
        print "før les",msecs
        off uart 2 input
        uart 2 read b
        if b==0x2 then
                let bytesread = 0
                while bytesread <= 10 do
                        uart 2 read b
```

```
                        if bytesread < 10 then
                                if b == 0x2 || b == 0xd || b == 0xa || b ==
                                break
                                endif
                                let idread[bytesread] = b
                        elseif bytesread == 10 then
                                uart 2 read b
                                uart 2 read b
                                uart 2 read b
                        endif
                        let bytesread = bytesread + 1
                endwhile
                print raw idread
        endif
        print "etter␣les,␣før␣send",msecs
        for i = 0 to idread#-1
                let rfid[i] = idread[i]
        next
        print "etter␣send",msecs
        on uart 2 input do gosub rxid
endsub
```

Tillegg G

Kode til på oppsett 5

G.1 Sentralenhet

```
dim test
dim aks[3]
while 1 do
    print test,aks
    sleep 100 ms
endwhile
```

G.2 Sensorenhet med akselerometer

```
dim an0 as pin an0 for analog input
dim an1 as pin an1 for analog input
dim an2 as pin an2 for analog input
dim aks[3] as remote on nodeid 2
while 1 do
    print msecs
    let aks[0] = an0
    let aks[1] = an1
    let aks[2] = an2
    print aks
endwhile
```

G.3 Sensorenhet 2

```
dim test as remote on nodeid 2
while 1 do
```

```
    let test = !test
    sleep 100 ms
endwhile
```

Tillegg H

Kode til på oppsett 6

H.1 Sentralenhet

```
dim test
dim aks[3]
while 1 do
    print test,aks
    sleep 100 ms
endwhile
```

H.2 Sensorenhet med akselerometer

```
dim an0 as pin an0 for analog input
dim an1 as pin an1 for analog input
dim an2 as pin an2 for analog input
dim aks[3] as remote on nodeid 1
while 1 do
    print msecs
    let aks[0] = an0
    let aks[1] = an1
    let aks[2] = an2
    print aks
endwhile
```

H.3 Sensorenhet 2

```
dim test as remote on nodeid 2
dim aks[3] as remote on nodeid 2
```

```
dim oldaks
while 1 do
   if aks[2]!=oldaks then
     let test = !test
     let aks[0] = aks[0]
     let aks[1] = aks[1]
     let aks[2] = aks[2]
     print msecs, aks, test
   endif
   let oldaks = aks[2]
endwhile
```

Tillegg I

Kode til på oppsett 7

I.1 Sentralenhet

```
dim rfid[10]
dim aks[3]
while 1 do
    print raw rfid,aks
    sleep 100 ms
endwhile
```

I.2 Sensorenhet med akselerometer

```
dim an0 as pin an0 for analog input
dim an1 as pin an1 for analog input
dim an2 as pin an2 for analog input
dim aks[3] as remote on nodeid 2
while 1 do
    print msecs
    let aks[0] = an0
    let aks[1] = an1
    let aks[2] = an2
    print aks
endwhile
```

I.3 Sensorenhet med RFID

```
dim b as byte
dim bytesread as byte
```

```
dim idread[10]
dim i
dim rfid[10] as remote on nodeid 2
configure uart 2 for 9600 baud 8 data no parity
on uart 2 input do gosub rxid
while 1 do
endwhile
sub rxid
        print "Before␣read:␣",msecs
        off uart 2 input
        uart 2 read b
        if b==0x2 then
                let bytesread = 0
                while bytesread <= 10 do
                        uart 2 read b
                        if bytesread < 10 then
                                if b == 0x2 || b == 0xd || b == 0xa || b ==
                                break
                                endif
                                let idread[bytesread] = b
                        elseif bytesread == 10 then
                                uart 2 read b
                                uart 2 read b
                                uart 2 read b
                        endif
                        let bytesread = bytesread + 1
                endwhile
                print raw idread
        endif
        print "After␣read,␣before␣send:␣",msecs
        for i = 0 to idread#-1
                let rfid[i] = idread[i]
        next
        print "After␣send:␣",msecs
        on uart 2 input do gosub rxid
endsub
```